VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Možnosti Unreal Engine 4

Possibilities of Unreal Engine 4

2017 Bc. Štěpán Guňa

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Štěpán Guňa**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Možnosti Unreal Enginu 4
Possibilities of Unreal Engine 4

Jazyk vypracování: čeština

Zásady pro vypracování:

Nová verze Unreal Enginu 4 sebou přináší pokročilé možnosti v oblasti vývoje her, vzdělávacích aplikací, vizualizací apod. To je velmi důležité zejména při simulací jevů a procesů, které nelze v reálnem světě jednoduše demonstrovat. Důvodem může být například to, že je to technicky nemožné nebo by přitom mohlo dojít k ohrožení zdraví skupiny nebo jedinců. Proto již dnes existují např. simulace operací v lékařství nebo simulace na úrovní molekulárního světa. Cílem této práce je zaměřit se na možnosti vizualizace v novém Unreal Engine 4. Práce se bude zabývat popisem a demonstrací vybraných části tohoto enginu a výsledky budou použity v aplikaci pro simulování reálného provozu virtuální jaderné elektrárny.

1. Nastudujte možnosti a techniky při návrhu a implementaci projektu pro realistickou vizualizaci v Unreal Engine 4.
2. V práci se zaměřte zejména na následující části:
a) Vizualizace map virtuálního světa.
b) Menu pro nastavení kvality a grafických možností.
c) Možnosti ovládání pohyblivých objektů ve scéně.
d) Fyzikální vazby mezi objekty a jejich interakce.
e) Projekce textur a vlastností na objekty (Decals).
3. Teoretické znalosti využijte k implementaci ukázkových příkladů pro zpracovávaná témata.
4. Příklady vhodně zakomponujte po domluvě s vedoucím do projektu pro vizualizaci jaderné elektrárny, popřípadě jiných vhodných projektů.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Martin Němec, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 20. dubna 2017

Rád bych na tomto místě poděkoval vedoucímu Ing. Martinu Němcovi, Ph.D. za jeho odbornou pomoc, cenné rady a čas, který mi věnoval během vypracování této práce. Dále bych rád poděkoval mým rodičům a blízkým za pomoc a podporu během studia.

Abstrakt

Tato diplomová práce se zabývá specifickými možnostmi herního enginu Unreal Engine 4 a jeho využití při vytváření aplikací zaměřených na vizualizaci a simulaci chování objektů reálného světa. V práci jsou popisovány možnosti a techniky použité při implementaci projektů pro realistickou vizualizaci a konkrétní možnosti Unreal Engine 4 pro řešení zadaných problémů.

Textová podoba práce je rozdělena do několika kapitol dle zadání, kde úvodem každé kapitoly jsou popisovány teoretické poznatky a možná řešení dané části. Následně jsou popisovány postupy použité při praktickém řešení.

Výstupem této práce jsou zásuvné moduly pro vizualizaci map virtuálního světa a nastavování grafických možností aplikací vytvořených v Unreal Engine 4. Dále byly vytvořeny ukázkové scény, na kterých jsou demonstrovány některé použité techniky při návrhu a implementaci projektů realistické vizualizace v Unreal Engine 4.

Klíčová slova: Unreal Engine, Herní engine, Vizualizace, Počítačová grafika.

Abstract

This master thesis deals with the specific possibilities of the Unreal Engine 4 and its usage in creating of applications focused on visualizing and simulating the behavior of real world objects. The master thesis describes the possibilities and techniques used in the implementation of projects for realistic visualization and the specific possibilities of Unreal Engine 4 for solving the submitted problems.

The text part of this work is divided into several chapters according to the assignment where the introduction of each chapter describes the theoretical knowledge and possible solution of the problem. The procedures used in the practical solution are also described.

The outputs of this work are plug-ins for visualization of maps and setting of the graphical possibilities of applications created in Unreal Engine 4. Also sample scenes have been created to demonstrate some of the techniques used in the design and implementation of realistic visualization in Unreal Engine 4.

Key Words: Unreal Engine, Game Engine, Visualization, Computer graphics

Obsah

Seznam použitých zkratek a symbolů

AO	–	Z angl. Ambient Occlusion, Okluze ambientního světla
API	–	Z angl. Application Programming Interface, Rozhraní pro programování aplikací.
FPS	–	Z angl. Frame Per Second, Hodnota představující počet zobrazených snímků za sekundu.
OS	–	Z angl. Operatting System, Operační systém
SSAO	–	Z angl. Screen Space Ambient Occlusion, Technika pro výpočet ambientní okluze scény v reálném čase.
UE	–	Unreal Engine
UI	–	Z angl. User Interface, Uživatelské rozhraní
UMG	–	Z angl. Unreal Motion Graphics, Editor uživatelského rozhraní v Unreal Engine.
VR	–	Virtuální realita

Seznam obrázků

Seznam tabulek

1 Úvod

Unreal Engine 4 je vývojový program pro vytváření interaktivních grafických aplikací, simulací, výukových programů a prezentačních nástrojů. Své uplatnění nachází především v herním průmyslu, ale lze jej využít i pro vizualizaci procesů, které není možné prakticky demonstrovat v reálném světě z důvodů ohrožení zdraví, technické neproveditelnosti nebo vysokých nákladů. Lze sem zařadit například simulace chování kolonií bakterií, simulace letů do vesmíru nebo aplikace pro výcvik obraných a bezpečnostních složek. Pomocí nových platforem pro virtuální realitu a Unreal Engine 4, který je vybaven jejich podporou, lze vizualizovaný proces pak ještě více přiblížit uživateli.

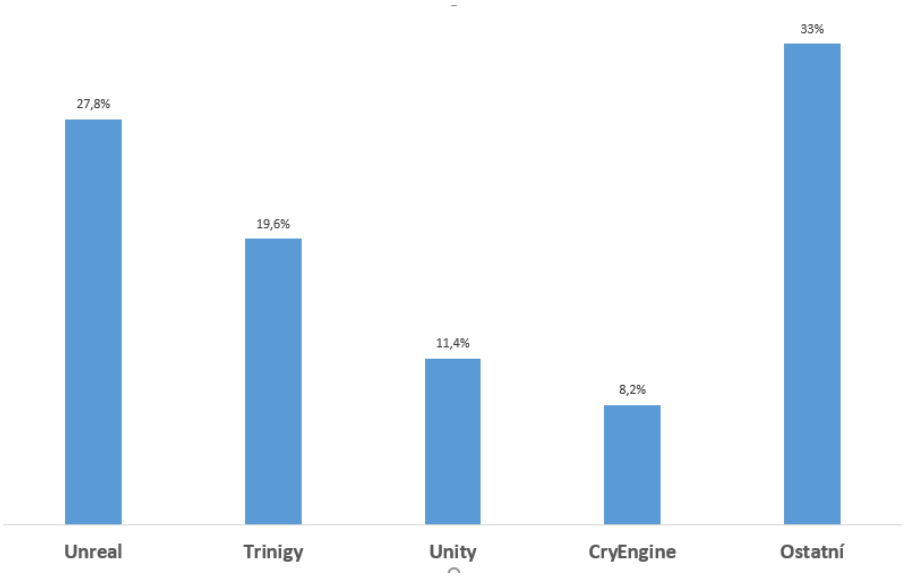
Procesem vytváření grafických aplikací založených na realistické vizualizaci se zabývá i tato práce. Primárním cílem této práce je obeznámit čtenáře s možnostmi a technikami Unreal Engine 4, které lze využít při vytváření těchto aplikací.

Textová forma práce je členěna do několika částí dle zadání, kde v první části jsou popisovány možnosti a nové zobrazovací techniky, které je možné využít při vývoji grafických aplikací v UE4. Tato část je zakončena popisem implementace a praktického využití metod popsaných v úvodu této části.

V dalších částech této práce jsou popisovány již konkrétní problémy vycházející ze zadání, jako je tvorba a vizualizace map virtuálního světa, tvorba modulu pro nastavování kvality a grafických možností, možnosti ovládání objektů a fyzikální vazby mezi objekty. Závěrem je pak popisována technika promítání textur na povrch objektů pomocí *Decals* projektoru. Součástí jednotlivých kapitol je vždy popis praktického řešení daného problému v prostředí UE4, kterému předchází teoretické nastínění a popis problematiky, včetně popisu důležitých metod a funkcí UE4 použitých při realizaci praktické části.

2 Přehled herních enginů

Kromě Unreal Engine 4, na jehož možnosti je zaměřena tato práce, jsou na trhu i další herní enginy spravované firmami zabývajícími se vývojem počítačových her a simulačních programů. Mezi tyto enginy patří například Unity, CryEngine, Source aj. Přehled popularity herních enginů v roce 2011 je zobrazen na obrázku 1.



Obrázek 1: Popularita herních enginů v roce 2011. [1]

Kromě nástrojů zaměřených na vývoj počítačových her existují i programy a vývojové nástroje zaměřené čistě na vizualizaci a renderování 3D scén. Nevýhodou těchto nástrojů ovšem je jejich striktní zaměření pouze na vizualizaci, popřípadě simulace, což zapříčiňuje absenci možností pro implementaci některých funkcionalit přímo ve vývojovém prostředí.

Herní enginy jsou pak do jisté míry universálním vývojovým nástrojem pro vytváření komplexních funkčních scén, vizualizací, simulací apod. Disponují škálou předdefinovaných funkcí, které nabízí uživateli k využití ve spojitosti s jeho modely, audio soubory, animacemi a jinými aktivy. Dále pak enginy obecně nabízí možnost implementace vlastních funkcionalit a modelů chování, zobrazovacích shaderů apod.

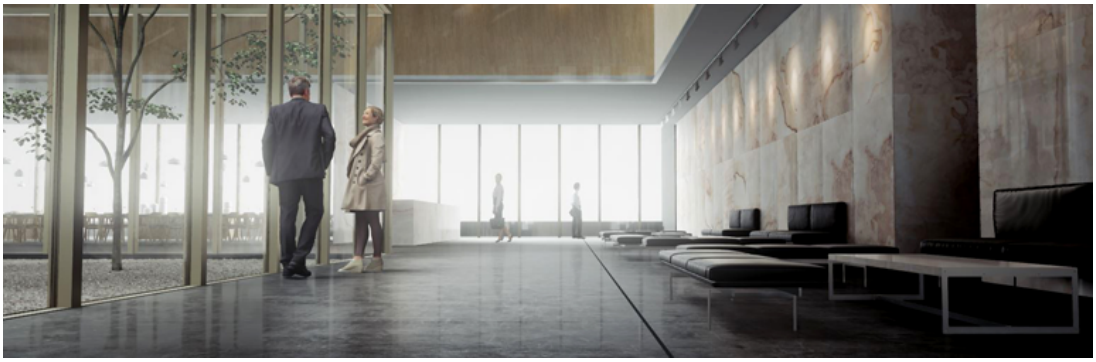
Díky své univerzálnosti nacházejí herní enginy kromě vývoje her uplatnění i v jiných odvětvích jako například v architektuře pro vizualizaci interiérů a exteriérů, pro tvorbu vizuálních efektů v kinematografii, zobrazování různých matematických a fyzikálních modelů, vytváření prezentačních materiálů, nebo nejrůznější simulace chování objektů, materiálů, lidí apod.

2.1 Unreal Engine

„Unreal Engine 4 je kompletní sada herních vývojových nástrojů, vytvořených vývojáři pro vývojáře.“ Takto popisují Unreal Engine 4 jeho tvůrci, společnost Epic Games [2], která herní engine vyvíjejí již od roku 1998. UE4 je sofistikovaný nástroj pro vývoj počítačových her, tvorbu animací, simulačních nástrojů a své uplatnění nalezne i v architektuře. Společně s Unity3D patří Unreal Engine 4 k nejrozšířenějším herním enginům. K jeho rozmachu napomohla volná licence, pod kterou je UE4 vydáván. Tímto krokem Epic Games reagoval na své konkurenty v oblasti herního vývoje. Využívání Unreal Engine 4 je tedy zcela zdarma, jedinou podmínkou, která vystává z licenční smlouvy a nahradila původní měsíční poplatek ve výši 19 $, je odevzdání 5% zisku z každé prodané aplikace vytvořené v UE4.

Unreal Engine je vydáván jako Open Source aplikace a jeho zdrojové kódy jsou volně k dispozici, což dává uživateli obrovskou volnost při vytváření aplikací. UE4 podporuje velkou škálu platforem a to včetně mobilních, konzolových a platforem pro VR. S novou verzí se objevil v UE4 i nový způsob vykreslování scén pomocí fyzikálně založeného renderingu. Dále engine disponuje spoustou vizuálních efektů, například adaptace oka při přechodu ze tmy do světla nebo záblesky v objektivu snímací kamery, a metod pro zpětné zpracování obrazu mezi které například patří SSAO pro výpočet ambientní okluze. Některé moderní zobrazovací metody používané v počítačové grafice budou popsány v následující kapitole věnované možnostem vizualizace pomocí UE4.

Komunita vývojářů UE4 dále spravuje podrobnou online dokumentaci [3] a provádí online workshopy s různými odborníky, díky čemuž má UE velmi dobrou uživatelskou podporu. Epic Games dále nabízí soustu aktiv (assetů) vytvořených přímo pro UE4 ve svém online obchodě. Ty pak lze na základě jednorázové platby volně využívat a urychlit tak vývoj projektu. Více informací o Unreal Engine a službách nabízených společností Epic Games lze nalézt na webových stránka viz [2]. Příklad scény vytvořené v prostředí Unreal Engine 4 je zobrazen na obrázku 2.

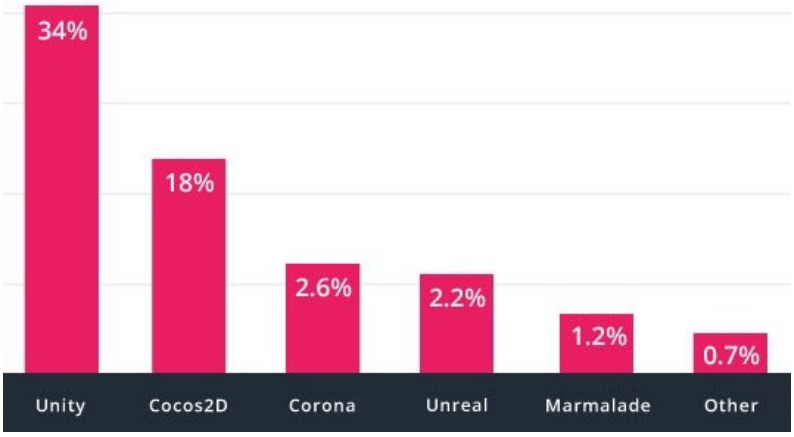


Obrázek 2: Příklad scény vytvořené v prostředí UE. [4]

2.2 Unity3D

Obrázek 3: Příklad scény vytvořené v prostředí Unity 5. [6]

Aktuálně se trhu vyskytuje verze Unity 5 a obdobně jako UE4 patří i Unity mezi nejpopulárnější herní enginy současnosti. Co se týká vývoje softwaru pro mobilní platformy Unity dokonce daleko předbíhá i UE a ostatní enginy (viz obrázek 4). To zapříčiňuje především snadné portování aplikací na mobilní zařízení, kterým Unity vyniká. Mezi přednosti Unity dále patří i velká škála podporovaných platforem, mezi které patří například i platforma SmartOS a VR. Mezi nevýhodu v porovnání s UE by se dala zařadit například absence „materiál editoru“ pro vytváření shaderů a nutnost dodatečně instalovat některé základní funkcionality, které jsou v UE standartně k dispozici. Více informací o herním enginu Unity3D je k dispozici na webových stránkách Unity [5], příklad scény renderované v prostředí unity je pak zobrazen na obrázku 3.



Obrázek 4: Herních enginy využívané pro vývoj mobilních aplikací v roce 2016. [6]

2.3 CryEngine

CryEngine [7] je jeden z velmi známých a oblíbených enginů v herním průmyslu. Vyvíjí jej německá společnost CryTek a do podvědomí běžných uživatelů se dostal především hrou Far Cry, ve které byl představen přelomový způsob grafického zpracování. Svou pozici na poli herních enginů upevnil dalším velmi známým herním titulem Crysis. V současnosti je na trhu verze CryEngine 5, která disponuje podporou VR a nově byl použit jazyk C# pro psaní skriptů. Příklad scény vytvořené v prostředí CryEngine 5 je zobrazen na obrázku 5. Více informací o CryEngine lze nalézt na webových stránkach společnosti CryTek [7].

Obrázek 5: Příklad scény vytvořené v prostředí CryEngine 5. [7]

2.4 Source

Je velmi známý herní engine od společnosti Valve a herní tituly vydané pod tímto enginem patří k těm nejznámějším v herním světě. Patří mezi ně například kolekce Half-Life, Left 4 Dead, a hry natolik úspěšné, že se uplatnily v celosvětových turnajích jako Counter Strike - Global Offensive a Dota 2. V současnosti je na trhu verze Source 2, kterou Valve poskytuje scela zdarma. Jedinou podmínkou, kterou si Valve Corporation klade v licenční smlouvě je, že hra vytvořená pod enginem musí být vydána i na platformě Steam [8] určené k digitální distribuci softwaru. Source 2 nově zahrnuje zobrazovací řetězec Vulcan API [10] a podporu platformy VR. Více informací o herním enginu Source 2 je k dispozici na webových stránkách Valve Comporation [9].

3 Realistická vizualizace v Unreal Engine 4

Realistická vizualizace je důležitou součástí při demonstraci vlastností různých objektů z reálného světa, jejich významu a interakcí. Jako taková se realistická vizualizace používá v mnoha odvětvích, například vizualizace interiérů a exteriérů v architektuře, ale taktéž při tvorbě nejrůznějších simulací a celé řady počítačových her. Účelem realistické vizualizace, je přiblížit se co nejvíce realitě a díky novým způsobům a metodám zpracování v grafických aplikacích se to i daří.

V rámci této kapitoly bude nastíněna metoda fyzikálně založeného renderování, kterou disponují moderní herní enginy a která je využita i při zobrazování scény v prostředí UE4. Následně bude popsán postup vytvoření modelů a jejich zpracování v UE, při implementaci projektu vizualizace jaderné elektrárny.

3.1 Fyzikálně založený rendering (PBR)

Fyzikálně založený rendering, je soubor zobrazovacích metod, které jsou založeny na stejných principech, na kterých fungují fyzikální zákony reálného světa. Konkrétně ty, které se týkají pohybu světla a chování jednotlivých fotonů při dopadu na povrch objektu s danými vlastnosti (odrazivost, rozptyl apod.). Výsledkem je pak reálnější dojem z vizualizované scény, než při použití klasických osvětlovacích algoritmů jako je Phong, Blinn - Phong apod.

Dle [11] je fyzikálně založený rendering popsán následující zobrazovací rovnicí:

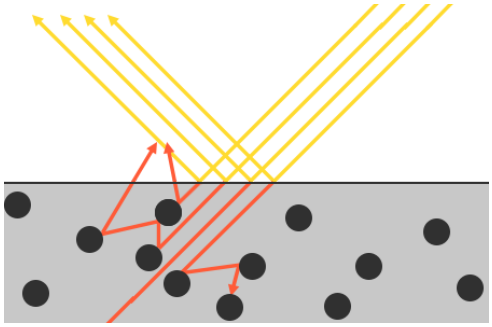
$$L_o\left(p,\omega_o\right) = \int_{\Omega} f_r\left(p,\omega_i,\omega_o\right) L_i\left(p,\omega_i\right)\left(\mathbf{n}\cdot\omega_i\right) d\omega_i, \tag{1}$$

která popisuje jakým způsobem je získávána hodnota světla vycházející z daného bodu scény (L_o) vzhledem k množství světla na bod dopadajícího (L_i); f_r je pak funkce popisující odraz světla daného bodu. L_o pak nese informaci o výsledné barvě.

Zákon o zachování energie: Fyzikálně založený rendering vychází ze zákonu o zachování energie, který popisuje jev, že odchozí světelná energie by nikdy neměla překročit vstupní světelnou energii (neplatí u ploch emitujících světlo). Ze zákonu o zachování energie tedy vyplývá:

$$\forall\omega_i \int_{\Omega} f_r\left(p,\omega_i,\omega_o\right)\left(\mathbf{n}\cdot\omega_i\right) d\omega_o \leq 1.$$

Odraz a rozptyl světla: Při dopadu světla na plochu dochází následně ke dvěma jevům. Prvním je zrcadlový odraz, kdy jsou paprsky odraženy podle normály dané plochy zpět do prostoru. Druhým jevem je lom světla směrem do prostoru plochy (viz obrázek 6). V tom případě jsou některé paprsky světla pohlceny materiálem a ostatní jsou odraženy náhodně zpět do prostoru (difuzní odraz). Na základě vlnové délky paprsků světla, které jsou materiálem vraceny zpět a které naopak materiál zcela pohltí, lze pak definovat vnímanou barvu materiálu.



Obrázek 6: Příklad reflexe a difuze dopadaného světla na povrch. [12]

3.1.1 Teorie mikro-ploch (Microfacets)

Všechny techniky PBR jsou zakládány na tzv. teorii mikro-ploch (Microfacets) [12], která popisuje, že jakýkoliv povrch v mikroskopickém měřítku může být popsán drobnými dokonale odrazivými zrcadly nazvanými Microfacets (Mikro-plochy).[13] Ilustrace mikro ploch a jejich vliv na dopadající světlo je zobrazena na obrázku 7.

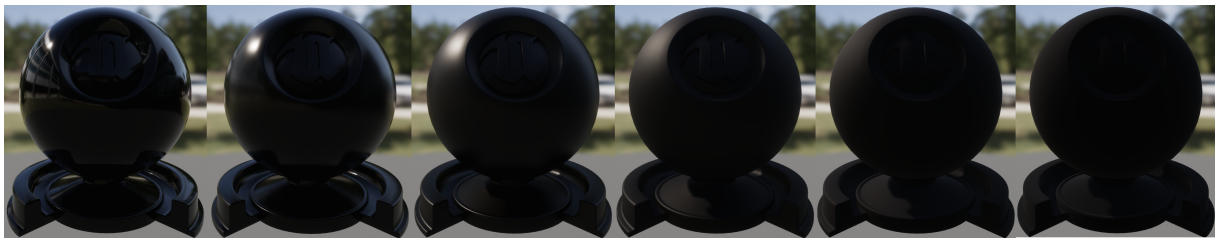
Obrázek 7: Vliv mikro ploch hrubého a hladkého materiálu na dopadající paprsky světla.[12]

Vzhledem k tomu, že mikro-plochy jsou natolik malé, že je není možné rozlišit na úrovni pixelů, je hrubost povrchu odhadnuta pomocí NDF (Normální distribuční funkce) na základě parametru hrubosti (Roughness). Příklad změny parametru hrubosti zobrazovaného materiálu je zobrazen na obrázku 8.Na základě tohoto parametru je pak možné určit poměr ploch, které jsou přibližně srovnány s vektorem **h**. Distribuční funkce normálního rozložení [14] je pak dána jako:

$$D(h) = \frac{\alpha^2}{\pi\left((\mathbf{n}\cdot\mathbf{h})^2(\alpha^2-1)+1\right)^2}, \tag{2}$$

kde vektor **h** je poloviční vektor mezi vektorem dopadajícího světla **l** a pohledovým vektorem **v**, kde:

$$\mathbf{h} = \frac{\mathbf{l} + \mathbf{v}}{\|\mathbf{l} + \mathbf{v}\|}. \tag{3}$$



Obrázek 8: Vliv změny parametru hrubosti na výsledný materiál.

3.1.2 Cook-Torrance BRDF

Bidirectional Reflectance Distribution Function (BRDF), česky jako obousměrná distribuční funkce odrazu světla. Tato funkce je v zobrazovací rovnici (1) použita pro získání hodnoty f_r, s ohledem na 1zrcadlový a difuzní odraz a zvolený materiál. BRDF funkce pro f_r je dle [12]:

$$f_r = k_d f_{lambert} + k_s f_{cook-torrance}, \tag{4}$$

kde $f_{lambert}$ popisuje difuzní složku a $f_{cook-torrance}$ složku zrcadlovou. Hodnoty k_d a k_s definují množství jednotlivých složek odrazu, kde ze zákona o zachování energie lze odvodit vztah:

$$k_d + k_s \leq 1$$

. Lambertovu funkci pro popis difuzní složky odrazu lze pak zapsat následovně:

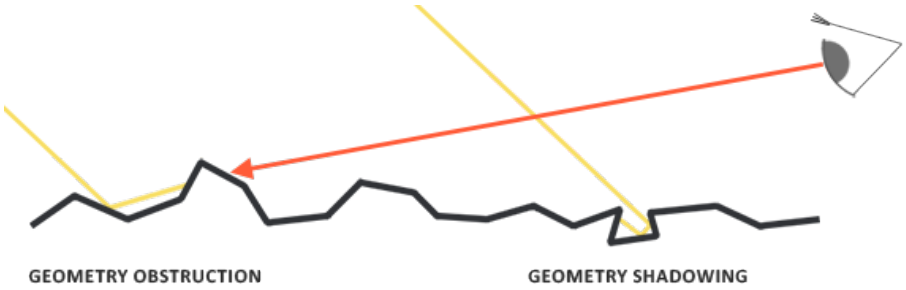
$$f_{lambert} = \frac{c}{\pi}, \tag{5}$$

kde c označuje tzv. albedo (barva povrchu). Cook-Torrance BRDF je pak dána jako:

$$f_{cook-torrance} = \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)}, \tag{6}$$

kde D označuje distribuční funkci která je popsána v 3.1.1. Funkce F představuje Fresnelovu funkci, která popisuje poměr odraženého světla a světla lomeného, který se mění v závislosti na úhlu vektoru pohledu. (Fresnelův efekt) Funkce G pak představuje geometrickou funkci, která popisuje zastínění povrchu jednotlivých mikro-ploch ostatními mikro-plochami. Hodnota zastínění je obdobně jako u NDF odhadována na základě zvolené hodnoty hrubosti povrchu.

Definici geometrické funkce lze najít v [12]. Příklad zastínění jednotlivých mikro ploch ostatními mikro plochami je ilustrován na nesledujícím obrázku (Obrázek 9).



Obrázek 9: Příklad zastínění jednotlivých mikro-ploch ostatními mikro-plochami.[12]

Po dosazení BRDF funkcí do zobrazovací rovnice (1) a její úpravě lze rovnici dle [11] přepsat následovně:

$$L_o\left(p,\omega_o\right)=\int_{\Omega}\left(k_d\frac{c}{\pi}+k_s\frac{DFG}{4(\omega_o\cdot n)(\omega_i\cdot n)}\right)L_i\left(p,\omega_i\right)\left(\mathbf{n}\cdot\omega_i\right)d\omega_i, \tag{7}$$

V rámci popisu PBR bylo čerpáno z [12] a [11], kde je pro vysvětlení principu využito obecných modelů. Popis konkrétních modelů použitých pro definování zobrazovací rovnice přímo v prostředí UE je k dispozici v [14].

3.2 Fyzikálně založené materiály v UE

V prostředí UE lze vytvářet různé druhy materiálů (shaderů), které lze rozdělit podle typu jejich využití:

Surface - Klasický materiál definující povrch nějakého objektu.

Deffered Decal - Typ materiálu používaného při promítání textury na povrch nějakého objektu. (viz kapitola 8)

Light Function - Materiály používané například pro zobrazování kužele světla.

Post Process - Speciální materiály, které se aplikují až na výsledný obraz scény.

Dále lze materiálům definovat, jaký model bude použitý pro jejich stínování (Shading Models). V UE4 jsou k dispozici tyto základní modely stínování:

Default Lit - Výchozí model, který do výpočtu zahrnuje i výsledky přímého a nepřímého osvětlení.

Unlit - Definuje konstantní stínování. Výsledná barva není ovlivněna použitým osvětlovacím modelem.

Surface - Model stínování definovaný pro průsvitné materiály. Například vosk svíčky, který propouští část světla a to se pak rozptyluje uvnitř materiálu. Na základě tohoto stínování lze v materiálu definovat barvu pod jeho povrchem.

Preintegrated Skin - Model určený pro vykreslování lidské kůže. Obdobný jako předchozí model. Výpočet ovšem není prováděn pomocí fyzikálního modelu, čemuž odpovídá i nižší náročnost na výpočrtní výkon.

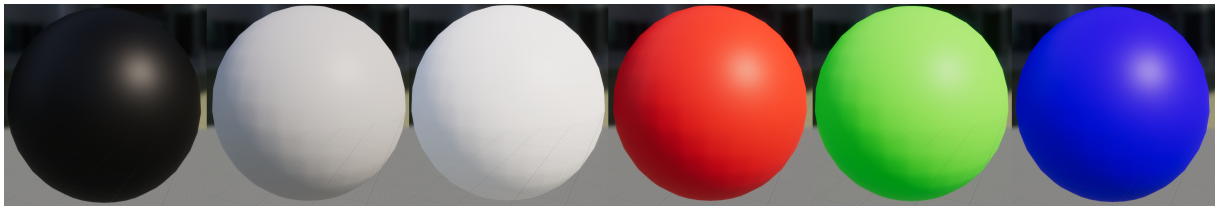
Clear Coat - Model pro simulaci vícevrstvých materiálů, které mají tenkou průhlednou vrstvu na povrchu klasického materiálu. Například bezbarvý lak na dřevě.

V reálném světe se vyskytují 3 druhy materiálů: izolanty, polovodiče a vodiče. Při vizualizaci jsou důležité především materiály dielektrické (izolanty) a kovové (vodiče). Toto rozdělení materiálů je důležité především pro PBR, kde na základě zvoleného typu materiálu jsou ovlivněny i výsledky funkcí D,F a H použitých při výpočtu BRDF (viz 3.1).

Při vytváření materiálů v prostředí UE, jsou pro definování jejich fyzikálních vlastností využity 4 základní parametry: Základní barva, kovovost, hrubost a odrazivost. Dalším parametrem, který do jisté míry ovlivňuje fyzikální vlastnosti materiálů je normála (Normálová textura), která provádí optickou deformaci povrchu a je využita pro definici míry nerovnosti (Roughness).

3.2.1 Základní barva (Base Color)

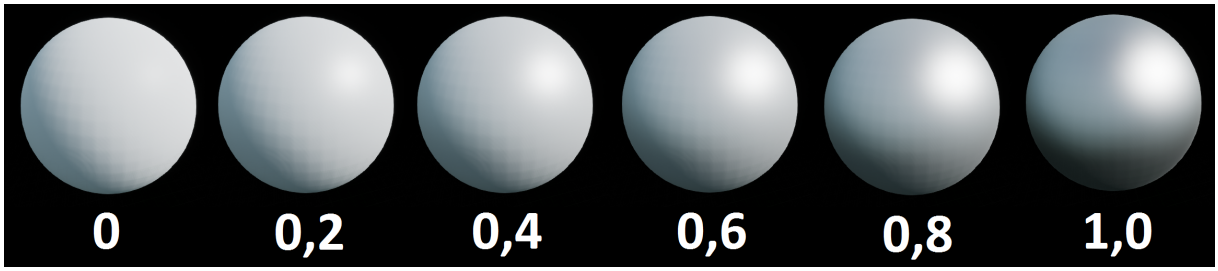
Parametr Base Color u nastavovaného materiálu udává hodnotu základní barvy materiálu (Albedo). Ta je v závislosti na ostatních parametrech modifikována v závislosti na výpočtu BRDF funkcí v PBR. Příklad různých nastavení parametru Base Color je zobrazen na obrázku 10.



Obrázek 10: Různé nastavení základní barvy materiálu.

3.2.2 Kovovost (Metalic)

Tento parametr nastavuje hodnotu poměru dielektrických a vodivých částí obsažených v materiálu na jeden zobrazovaný pixel. Příklad různých nastavení parametru Metalic je zobrazen na obrázku 11.



Obrázek 11: Různé nastavení poměru vodivých a dielektrických materiálu (Kovovost).

3.2.3 Hrubost (Roughness)

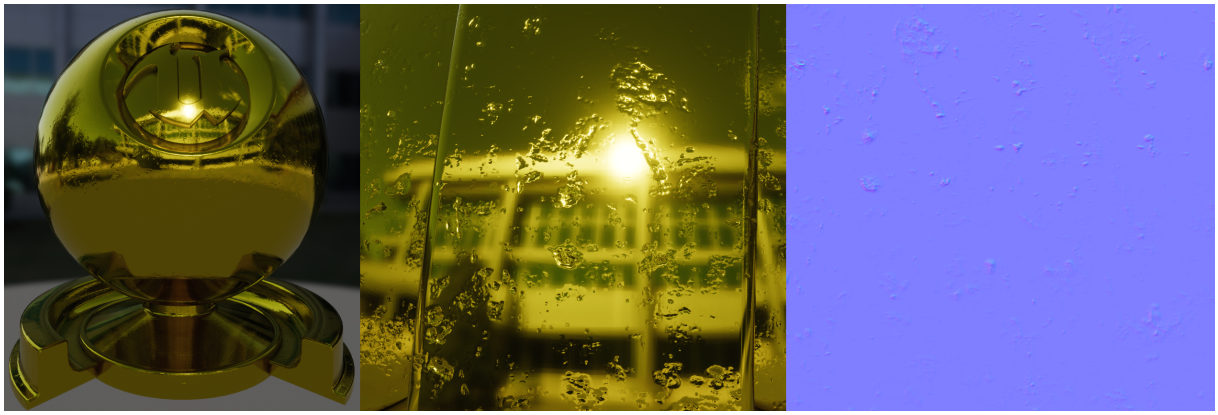
Tento parametr slouží pro nastavení hrubosti povrchu a nastavuje způsob jakým jsou odráženy paprsky dopadající na povrch. Příklad nastavení různých hodnot hrubosti materiálu byl ukázán na obrázku 8.

3.2.4 Odrazivost (Specular)

Provádí nastavení odrazivosti výhradně u dielektrických materiálů. U kovových materiálů nemá z fyzikálního hlediska žádné opodstatnění.

3.2.5 Normálová mapa

Normálová mapa do jisté míry ovlivňuje hrubost povrchu na základě optické změny. Hodnot uložené v normálové mapě lze použít pro nastavování parametru hrubosti materiálu. Příklad využití normálové mapy pro nastavení parametru Roughness je zobrazen na následujícím obrázku 13.



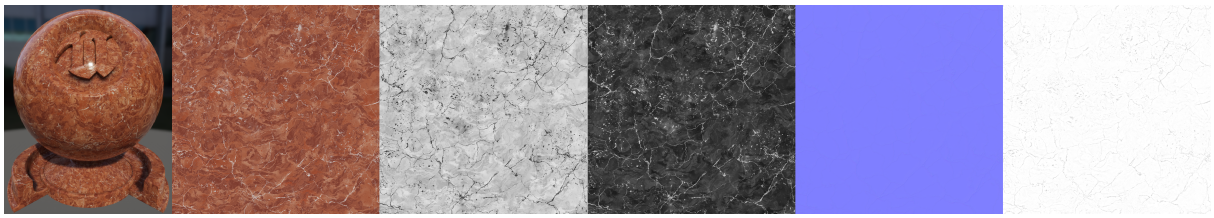
Obrázek 12: Příklad použití hodnot z normálové mapy pro nastavení hrubosti materiálu.

3.2.6 Ambientní okluze

Parametr Ambient Occlusion slouží u PBR materiálů k nastavení zastínění jednotlivých mikroploch jejich okolím. Tento parametr ovlivňuje výstup geometrické funkce použité u BRDF při výpočtu odráženého světla (viz 3.1). Textury pro AO jsou většinou vytvářeny manuálně na základě topologie zobrazovaného objektu nebo objektů scény.

3.2.7 Použití textur

Jednotlivé parametry lze také nastavit na základě hodnot obsažených v textuře. Pro PBR a fyzikálně založené materiály se používají speciální textury (PBR textury), které obsahují informace a nastavení určené pro zobrazování pomocí fyzikálně založeného renderování.



Obrázek 13: Použití PBR textur: Albedo, Metalic, Roughness, Normal, AO

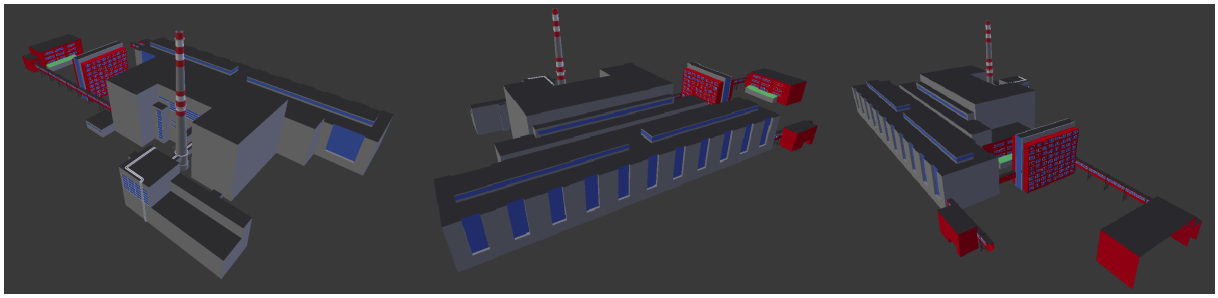
3.3 Praktické využití prostředí Unreal Engine 4

V rámci praktické části diplomové práce bylo vytvořeno několik modelů a scén, které byly následně pomocí UE4 vizualizovány v projektu virtuální jaderné elektrárny. Fotky z jednotlivých scén jsou k dispozici v příloze této páce (viz A - E). Na základě vytvořených modelů byla sestavena také ukázková aplikace, na které bylo demonstrováno použití UE4 pro realistickou vizualizaci. Tato aplikace je součástí elektronické přílohy.

3.3.1 Vytvoření a příprava modelů

Pro projekt vizualizace provozu jaderné elektrárny byly v modelovacím nástroji Blender [15] vytvořeny modely exteriéru a interiéru budov sloužící provozu JE. Jednotlivé modely byly vytvářeny na základě fotodokumentace v poměru 1:1 podle reálných budov. Dále byly modelům vytvořeny tzv. kolizní objekty, které jsou použity v UE pro výčty kolizí daného objektu s okolními objekty. U modelů s jednoduchou topologií byly kolizní objekty generovány automaticky až po vložení modelů do prostředí UE. V Blenderu byly k některým objektům vytvořeny i objekty s méně detailnější strukturou, tak aby je bylo možně použít jako statické LODy v rámci snižováni nároků na renderování výsledné scény v UE.

Vytvořené modely byly následně vyexportovány jako FBX soubory a vloženy do prostředí enginu. Příklad modelů budov primárního a sekundárního okruhu JE, vytvořených v modelovacím nástroji Blender je zobrazen na obrázku 14



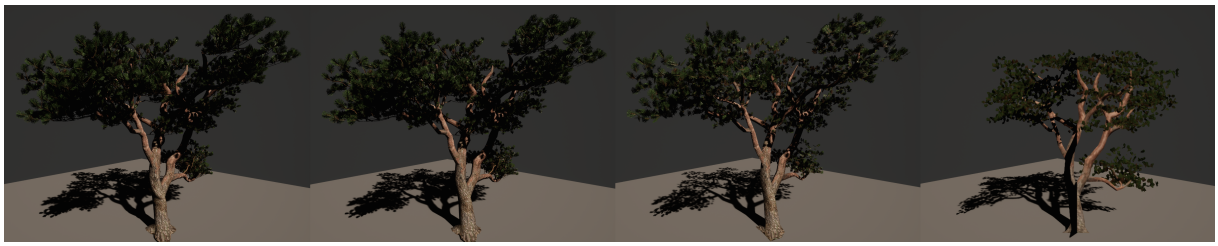
Obrázek 14: Příklad vytvořeného modelu v programu Blender.

Při vytváření některých modelů byly k daným objektům vytvořeny i modely s menší úrovní detailů (tzv. LODy). Přepínání mezi jednotlivými modely, představující vizualizovaný objekt se pak provádí na základě vzdálenosti pozorovatele (kamery snímající scénu) od daného objektu. Čím více se vzdaluje kamera od objektu je pro jeho zobrazení používán model s menší úrovní detailů.

Statické LODy: Jedná se o uspořádanou množinu předem vytvořených modelů, ze které se při zobrazování ve scéně vybírá model, odpovídající vhodným kritériím (Vzdálenost vykreslovaného objektu od snímající kamery). Výhodou statických LODů je především menší náročnost na jejich režii a výpočet, protože modely jsou předem vytvořeny a uloženy do paměti. Nevýhodou statických LODů je tzv. Popping efekt, který se projevuje při přepínaní jednotlivých modelů. Tomuto jevu se dá částečně zabránit pomocí některých modifikací (např.: mícháním sousedních úrovní - v UE od verze 4.9).

Dynamické LODy: Se zásadně liší od statických v tom, že model s danou úrovní detailů se vytváří přímo za běhu aplikace s každým renderovaným snímkem. To zvyšuje výpočetní nároky, ale účině eliminuje tzv. popping efekt. Další výhodou jodou ideální úroveň detailu modelu v každém renderovaném snímku a menší paměťové nároky. Typickým příkladem algoritmu výpočtu dynamických lodů jsou tzv. progresivní trojúhelníkové sítě [16].

S novou verzí UE 4.15 byla do enginu přidána možnosti generovat jednotlivé LODy automaticky přímo v enginu, kterou lze použít namísto manuálního vytváření jednotlivých modelů. Příklad modelů s různou úrovní detailů ve scéně UE je zobrazeno na obrázku 15.

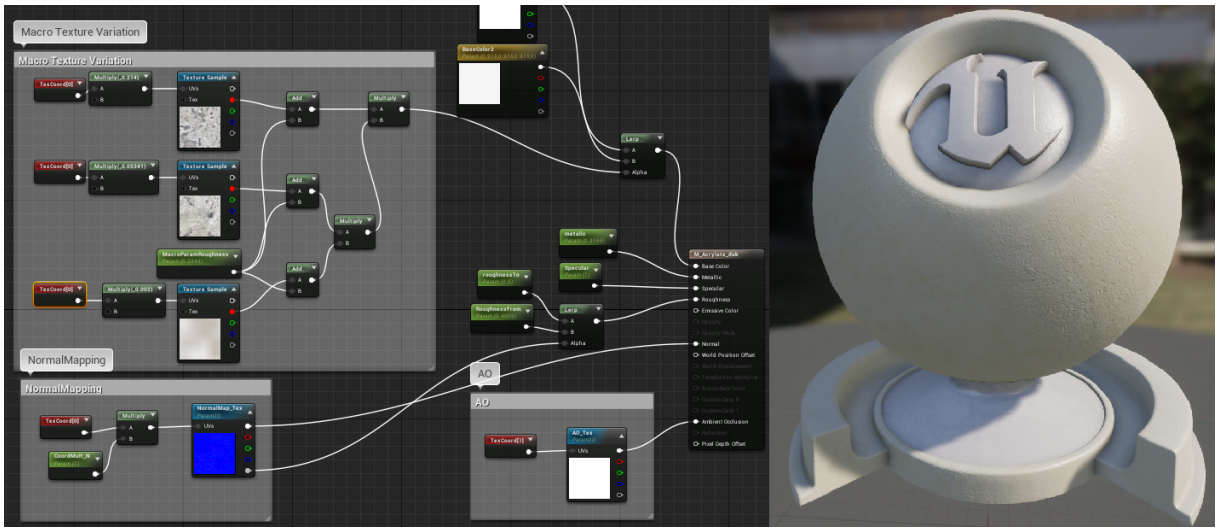


Obrázek 15: Různé úrovně detailů modelu.

3.3.2 Vytváření materiálů

Po importování vytvořených modelů do prostředí Unreal Engine bylo potřeba vytvořit zobrazovací shadery (materiály), ve kterých je počítána výsledná barva zobrazovaného objektu ve scéně. UE4 disponuje uživatelsky příjemným nástrojem pro vytváření shaderů tzv. materiál editorem, ve kterém lze pomocí UI definovat různé druhy zobrazovacích shaderů.

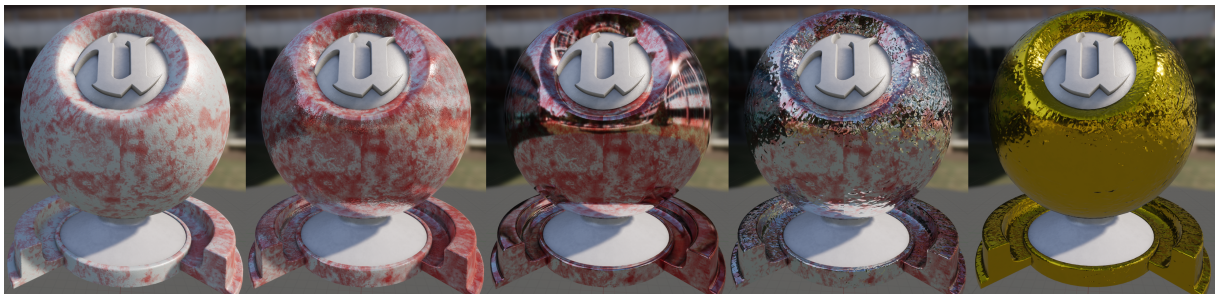
Akrylátový nátěr: Příklad materiálu vytvořeného pro použití na kovových modelech je zobrazen na obrázku 16. Tento referenční materiál byl parametrizován a za pomocí instancí z něj vytvořených byl upravován a použit na většině „metalických“ modelech.



Obrázek 16: Příklad parametrizovaného materiálu.

3.3.3 Využití instancí materiálu:

Během práce s prostředím UE4 se ukázalo výhodnější, především z časových důvodů, využívat instance materiálů. Pro to bylo vytvořeno několik referenčních shaderů pro určitý druh materiálu (omítka, barva, dřevo apod.). Na základě těchto shaderů bylo možné vytvářet jejich instance a pomocí parametrů upravovat některé detaily a vlastnosti. (například odrazivost, hrubost, použitou texturu aj.) Používáním instancí referenčních materiálů pak ulehčilo práci se scénou, neboť změny provedené v instanci se okamžitě projevují na modelu ve scéně bez nutnosti materiál znovu kompilovat a sestavovat. Příklad několika instancí materiálu z obrázku 16 je zobrazen na následujícím snímku 17.



Obrázek 17: Instance materiálu zobrazeného na obrázku 16

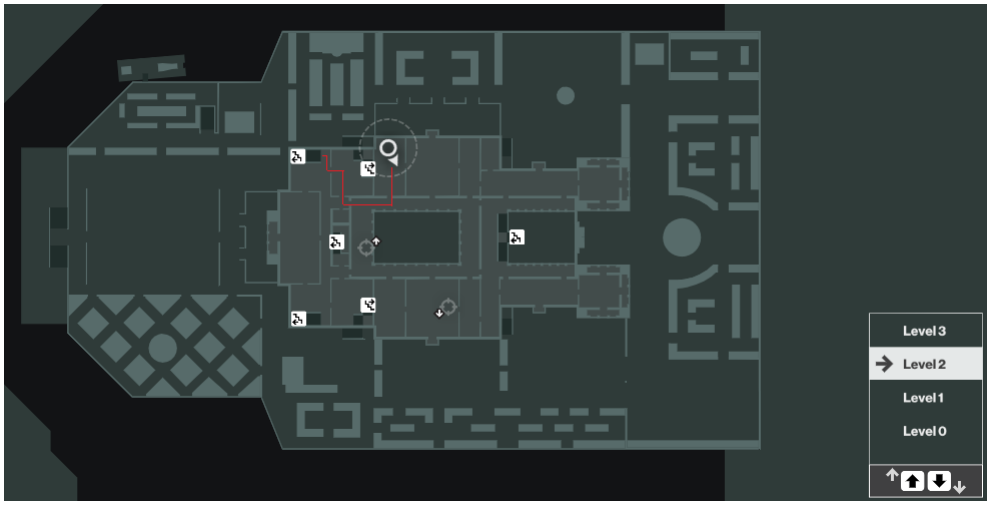
4 Vizualizace map virtuálního světa

Mapy virtuálního světa slouží pro zjednodušení orientace uživatele ve vizualizované scéně a jeho navigaci ve virtuálním světě. Především je-li virtuální svět příliš obsáhlý je potřeba jeho podobu nějakým způsobem zaznamenat a vhodným způsobem uživateli virtuální svět přiblížit. Způsobů jak vytvářet mapy virtuálního světa existuje velké množství a jednotlivé způsoby vytváření a zobrazování map se odvíjí od daného typu použití. Například při vizualizaci interiérů je vhodné použit jiný způsob vytváření map než při vizualizaci otevřeného světa.

V této kapitole budou popsány nejpoužívanější způsoby tvorby map a způsoby navigace použité v některých známých herních titulech. Následně pak bude popisována implementace generátoru mapy virtuálního světa v UE4 s využitím „Scene Capture 2D“ komponenty.

4.1 Mapy, minimapy a radary

Pro vizualizaci mapy virtuálního světa se v počítačových hrách většinou využívá rastrových dat (obrázků), které reprezentují prostředí zobrazovaného světa. Možností jak tyto data získat a vizualizovat mapu virtuálního světa je hned několik. Velmi často se lze setkat s mapami již předkreslenými, do kterých se pak zanáší pozice hráče a významných cílů ve scéně (například mapa z počítačové hry Hitman 2016 zobrazena na obrázku 18). Obrázky, respektive textury, které představují mapu jsou pak v daném měřítku vůči zobrazovanému virtuálnímu světu nebo jeho části. Obrázek tvořící mapu lze také získat snímáním povrchu scény z ptačí perspektivy a zanášením výsledku do textury. Tohoto způsobu bylo využito při implementaci generátoru map v praktické části této práce.



Obrázek 18: Příklad vizualizace mapy ve hře Hitman 2016. [17]

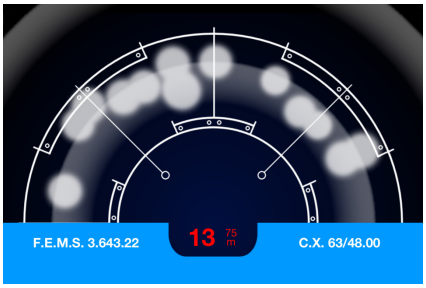
Do vygenerovaných nebo předkreslených obrázků jsou následně zaznamenávány pozice a to tak, že pozice objektu ve 3D scéně je převedena do prostoru 2D mapy. Z vektoru představujícího polohu zaznamenávaného objektu jsou brány většinou jen hodnoty x a y, které představují polohu vzhledem k rovině virtuálního světa. Hodnota z je pak využívána pro výpočet výšky polohy objektu. Například při zobrazování mapy interiéru je výšková souřadnice použita k přepínání mezi jednotlivými obrázky (půdorysy) v závislosti na patře ve kterém se uživatel nachází. Pomocí výškové souřadnice mohou být počítány i další údaje jako například nadmořská výška objektu ve scéně, určení výšky polohy objektu ve scéně vůči výšce uživatele apod.

$$u = x_{min} + \frac{x - x_{min}}{x_{max} - x_{min}} \cdot (u_{max} - u_{min}), \tag{8}$$

$$v = y_{min} + \frac{y - y_{min}}{y_{max} - y_{min}} \cdot (v_{max} - v_{max}), \tag{9}$$

Pro přepočet světových souřadnic objektu na souřadnice ve vizualizované mapě lze pomocí úměry použít následujících vztahů 8 a 9, kde $u \in \langle 0; u_{max} \rangle$ a $v \in \langle 0; v_{max} \rangle$ představují souřadnice v prostoru mapy. Hodnoty x a y pak představují souřadnice objektu ve scéně umístěného v oblasti promítací roviny snímací kamery, která je dána hodnotami $x_{min}, y_{min}, x_{max}, y_{max}$.

Dalším způsobem, jak uživateli ulehčit orientaci ve virtuálním prostředí je pomocí radaru, na kterém se zobrazuje pozice blízkých objektů v závislosti na vzdálenosti od pozice uživatele. Na radaru je zobrazováno jen nejbližší okolí scény. Uživatel tak získá přehled o tom co se ve scéně děje kolem něj, ale nevytvoří si představu o celém virtuálním světě. Využití radaru jako navigačního prostředku je proto vhodné použít u malých scén a virtuálních světů, popřípadě jako prostředek pro doplnění informací z mapy celého světa.



Obrázek 19: Příklad vizualizace radaru ve hře Alien vs Predator. [18]

Radar jako jediný způsob navigace byl použit například ve hře Alien vs. Predátor, kde v kampani za mariňáka byl uživateli k dispozici pro navigaci po světě pouze senzor pohybu, na kterém se zobrazovali pohyblivé objekty kolem hráče (viz obrázek 19). Cílem toho bylo navodit hororovou atmosféru. Implementace radaru je obdobná jako u klasické mapy. objekty zobrazované na radaru jsou vždy v určité vzdálenosti (radius) od uživatele a objekty vzdálenější

do radaru zanášeny nejsou. Pozici objektu zaznamenávaného do radaru je opět potřeba převést ze světových souřadnic do prostoru mapy (radaru), k čemuž je možné opět využít vztahů 8 a 9.

4.2 Tvorba mapy pomocí „Scene Capture 2D“ komponenty

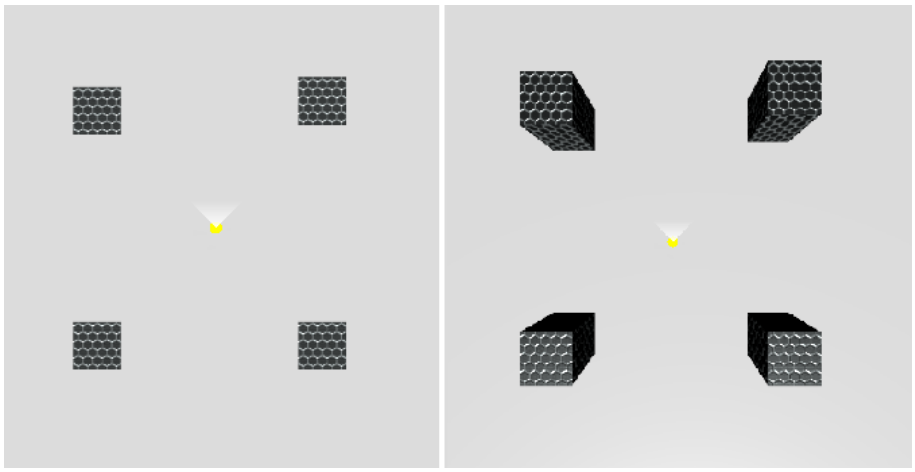
Jedním ze způsobů jak zachytit obraz virtuálního světa v podobě mapy je snímáním povrchu scény z ptačí perspektivy. Pro tuto možnost byla při implementaci map využita komponenta *Scene Capture 2D Component*. Tato komponenta v UE4 slouží ke snímání scény pomocí další kamery. Výstupem této kamery, umístěné ve scéně je speciální 2D textura v UE4 označována jako *Capture Target*. Tato textura je proměnná v čase, respektive její obsah se mění s každým renderovaným snímkem. Textura obsahuje obrázek právě snímané scény z pozice a s nastavením daným *Scene Capture 2D* komponentou. Vytvořenou texturu je pak možné použít v materiálu, nebo zobrazit pomocí uživatelského rozhraní na obrazovku uživateli. Při implementaci generátoru mapy virtuálního světa byly do obrázku zanášeny značky představující pozice hráčů a objektů ve scéně.

4.2.1 Možnosti Scene Capture 2D

Scene Capture 2D má spoustu nastavení, kterými lze ovlivňovat výsledek renderovaného snímku obdobně jako u klasické kamery použité pro renderování scény. Lze provádět korekce barev například vyvážení bíle barvy, natavení kontrastu, saturace, gama korekce, nebo nastavovat ambientní okluze, globální osvětlování apod. V nastavení Scene Capture 2D lze definovat i aktory scény, kteří nemají být zahrnuti do renderování (možnost Hidden Actors), popřípadě lze definovat pole aktorů ze scény, kteří budou jako jediní rendrováni (možnost Show Only Actors). Dále lze v nastavení definovat které post-process efekty použité ve scéně budou renderovány pomocí Scene Capture (možnosti v Advanced Show Flags).

Nutno podotknout, že většina těchto možností nastavení byla do UE4 přidána až po verzi 4.12. a v předchozích verzích, ve kterých byly implementovány i praktické části této práce, bylo nutné jednotlivé možnosti nastavení implementovat přímo ve zdrojovém kódu komponenty, popřípadě bylo potřeba vytvořit rozšíření, které chybějící možnosti nastavení doplňovali. (Možnosti které nebylo možno standardně nastavit přímo v detail panelu komponenty jsou například: Způsob promítání, definice aktorů, kteří nebudou renderováni, možnost renderovat jen základní barvy objektů apod.).

Důležitým nastavením je pak samotné nastavení promítací roviny a způsobu promítání. Při renderování snímku pro mapu scény bylo potřeba nastavit kameru na orthografické (rovnoběžné) promítání. Perspektivní promítání způsobovalo značné zkreslení a nebylo vhodné pro zobrazování mapy. Vzhledem k tomu, že nastavení promítací roviny nebylo do verze UE 4.12 součástí Scene Capture 2D komponenty bylo potřeba pro rovnoběžné promítání snímané scény změnit projekční matici přímo ve zdrojovém kódu (S novou verzí UE4 byl aktualizován i generátor map tak, aby byly využity metody nastavení implementované vývojáři UE). Rozdíl mezi použitým způsobem promítání u kamery snímající scénu z ptačí perspektivy je zobrazen na obrázku 20, kde nalevo je použito rovnoběžné a napravo perspektivní promítáni.



Obrázek 20: Způsoby promítání - zkreslení zobrazované mapy: rovnoběžné(vlevo) perspektivní(vpravo)

4.3 Praktické využití Scene Capture 2D pro generování map

V rámci implementace generátoru map procházené scény byla vytvořena třída *LevelSceneMapBP*, jejíž instance (Actor) je umísťována do scény. Aktor pak využívá Scene Capture 2D komponentu pro snímání procházené scény. Třída *LevelSceneMapBP* byla vytvořena tak, aby na základě interakce od uživatele generovala uživatelské rozhraní, ve kterém je zobrazován zachycený snímek scény z ptačí perspektivy. Do uživatelského rozhraní mapy jsou pak následně dokreslovány pozice hráče, pozice spoluhráčů a pozice významných objektů, které jsou generátoru předávány pomocí parametrů. Generátor pro vytváření a zobrazování map virtuálního světa byl implementován tak, aby jej bylo možné použít i v jiných projektech pod UE4. Jednotlivé parametry pro nastavení generátoru lze pak nastavit přímo v detail panelu instance (aktora) umístěné ve scéně.

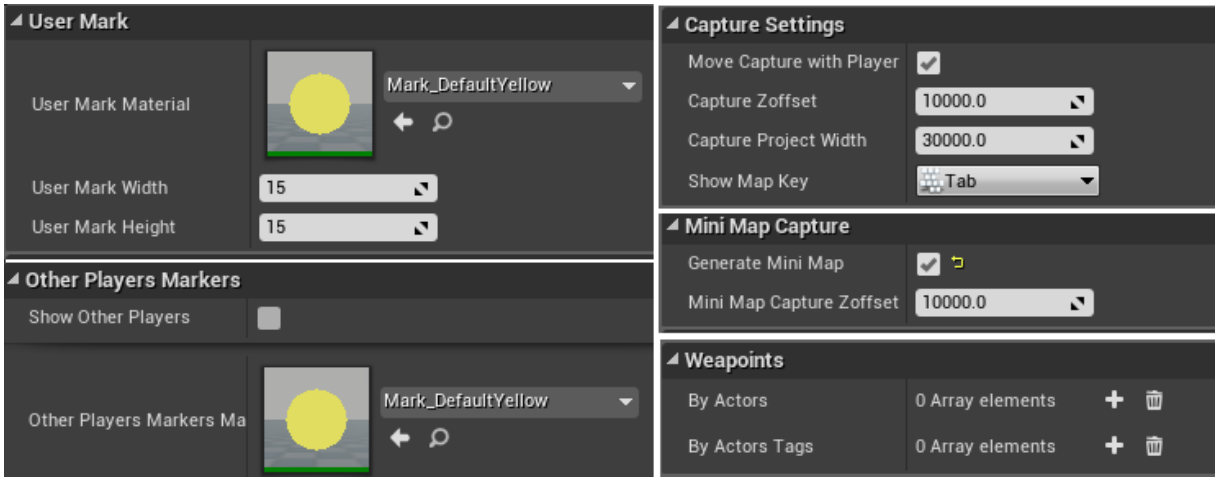
4.3.1 Nastavení generátoru

Třída *Level Scene Map BP*, dědí vlastnosti, metody a proměnné z třídy Scene Capture 2D Component. Parametry zděděné ze Scene Capture a označené jako editovatelné (Editable - označuje proměnné, které lze nastavit dané instanci z instance jiné třídy) je pak možné nastavovat i u instance třídy *Level Scene Map BP*, stejně jako kdyby byla do scény vložena samostatná Scene Capture 2D komponenta. V konstruktoru vytvořené třídy generátoru jsou ovšem nastaveny některé parametry Scene Capture komponenty tak, aby je uživatel již měnit nemohl. Parametry převzaté ze Scene Capture 2D a důležité pro nastavení generátoru jsou následující:

Capture Target - označuje texturu, do které se budou ukládat výsledky renderování. Tato textura slouží jako obrázek mapy a v konstruktoru je hodnota nastavována automaticky.
Projection Mode - definuje způsob promítání. V konstruktoru je výchozí hodnota nastavena na orthografické a uživatel toto nastavení nemůže měnit.
Hidden Actors - umožňuje nastavit pole aktorů ze scény, kteří nebudou renderování do mapy.
Show Only Actors - umožňuje nastavit pole aktorů ze scény, kteří budou jako jediní renderování do mapy.

Pomocí dalších parametrů, které jsou vlastní třídě *Scene Capture 2D Component* a jejich výchozí hodnotu nastavuje její konstruktor, lze nastavovat korekce výsledných barev, post-process efektů výpočtu osvětlení aj. Všechna tato nastavení pak mají vliv na výslednou barevnost a kvalitu renderovaného obrázku použitého pro zobrazování mapy.

V detail panelu instance třídy *Level Scene Map BP* vložené do scény pak lze nastavit další parametry, které jsou již vlastní třídě samotné a slouží pro definici značek zobrazovaných na mapě, některých vlastností promítání a vlastností mapy. Parametry jsou rozděleny do několika skupin podle jejich souvislosti (viz obrázek**??**), kde *Capture Settings* je sada parametrů pro dodatečné nastavení Scene Capture komponenty a vlastností mapy. Parametry jsou následující:



Obrázek 21: Detail panel a možnosti nastaveni generátoru map.

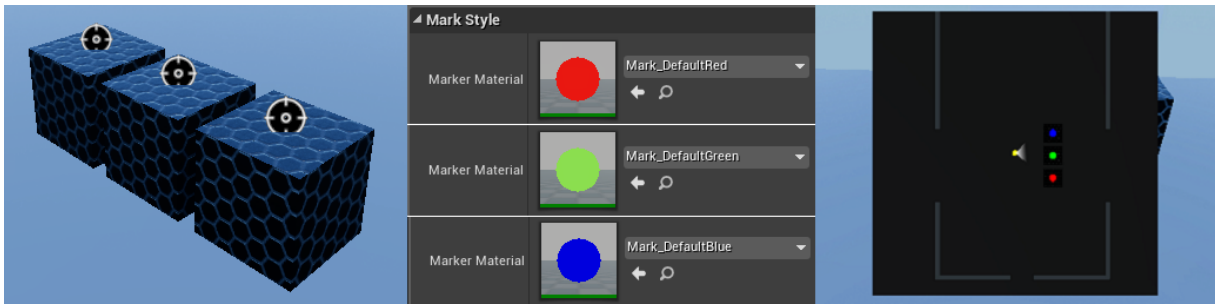
Move Capture With Player - umožňuje nastavit aby se kamera snímající povrch scény pohybovala současně s hráčem.
Capture Z offset - nastaví polohu snímací kamery v ose z. Objekty nacházející se ve scéně nad touto hranicí nebudou renderovány do obrázku mapy (budou nad kamerou).
Capture Projection Width - nastavuje velikost promítací roviny. Tato hodnota ovlivňuje jaký prostor scény bude zobrazován na mapě.
Show Map Key - tlačítko, kterým bude vyvoláno zobrazení mapy na obrazovce uživatele.

Sekce *Weapoints* pak slouží pro nastavení objektů, které se budou zobrazovat na mapě. Aktory lze definovat pomocí označení (tagů) v *By Actor Tags*, kde všechny objekty ve scéně, které mají dané označení budou zaneseny do mapy. Jednotlivým skupinám pak lze nastavit materiál, který bude použit pro jejich zobrazení na mapě (barva značky). Objekty které mají být zobrazovány na mapě lze definovat i jednotlivě v *By Actors*, kde lze přidávat jednotlivé reference na aktory ze scény do polí aktorů. Jednotlivým skupinám aktorů pak lze opět nastavit materiál použitý pro zakreslení značky. Materiál pro označení aktuálního hráče popřípadě ostatních hráčů lze nastavit v sekci *User Mark* a *Other Players Markers*.

Skupina parametrů v *Mini Map Capture* slouží pro generování minimapy. Pomocí *Mini Map Capture Zoffset* lze nastavit pozici snímací kamery v ose z podobně jako u mapy. Na minimapě se budou vykreslovat značky definované pro mapu v sekcích *User Mark*, *Other Players Markers* a *Weapoints*.

4.3.2 Třída Weapoint

Pro další způsob jak vkládat body do generované mapy byla implementována třída *Weapoint*, jejíž instance jsou vkládány do scény jako aktéři. Při vytváření jsou pak v generátoru procházeny všechny tyto instance umístěné ve scéně a pro každou z nich je vytvořena značka, která bude zobrazena na mapě. Jednotlivým instancím lze nastavit materiál, který bude použit při vytváření a zobrazování značky na mapě. Příklad použití vytvořené třídy ve scéně a následné zobrazení jednotlivých instancí na mapě je zobrazen na obrázku 22.

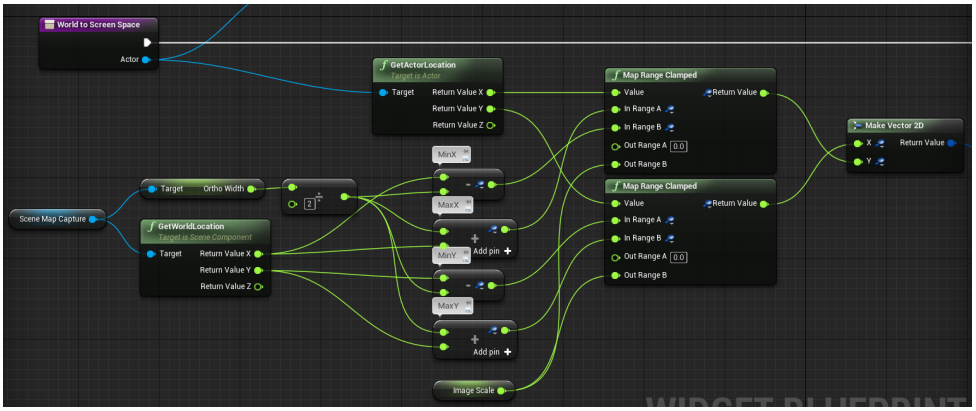


Obrázek 22: Příklad použití instancí třídy Weapoint ve scéně a zanesení značek do mapy.

4.3.3 Zpracování parametrů a generování mapy

Při zmáčknutí tlačítka pro zobrazení mapy, je na obrazovku uživatele zobrazováno uživatelské rozhraní s obrázkem, který představuje mapu virtuálního světa. Tento obrázek je renderován Scene Capture kamerou, která je kvůli úspoře výpočetního výkonu zapnuta až je li potřeba generovat mapu (zmáčknutí tlačítka uživatelem).

Před samotným vytvořením UI pro mapu je ve třídě *Level Scene Map BP* vytvořeno pole značek, které budou zaneseny do mapy (jedná se o pole referencí na vytvořené instance z třídy MarkUI). Při vytváření tohoto pole značek jsou procházeny jednotlivé pole referencí na objekty ve scéně předávány třídě *Level Scene Map BP* jako parametry. Do pole značek jsou přidány i značky jednotlivých uživatelů a značky vytvořené na základě všech instancí třídy *Weapoints* umístěných ve scéně. Pro získání všech značek, které mají být zaneseny do mapy byla v generátoru implementována metoda *Get Markers Array*, která vrací pole referencí na instance uživatelského rozhraní *Map Mark UI* (jde o widgety, které představují značku na mapě).



Obrázek 23: Výpočet pozice 3D objektu ve 2D mapě.

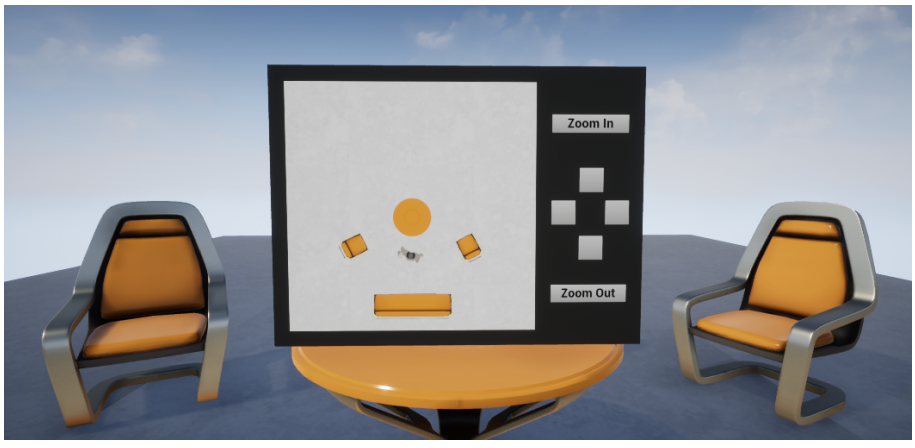
Vytvořenému uživatelskému rozhraní pro mapu jsou jako parametry předávány: pole značek a reference na kameru, která snímá povrch scény. Reference na kameru je v UI potřeba pro převod pozice objektu ve scéně na pozici značky objektu na mapě, který se provádí na pozadí UI. Funkce provádějící výpočet pozice značky na mapě je zobrazena na obrázku 23. Tato funkce zpracovává všechny reference na widgety představující značku na mapě a na základě pozice objektu ve scéně vypočítá polohu značky na mapě. Na pozadí UI pro mapu je dále prováděna funkce zajišťující, že se nebudou zobrazovat značky mimo mapu (funkce *CheckMapArea*, která vrací boolean hodnotu zda se značka nachází v prostoru mapy či nikoliv) a funkce pro aktualizaci pozice jednotlivých značek na mapě (funkce *Update Mark* a *UpdateUserMark*).

Sadu implementovaných struktur, tříd a widgetů lze použít jako plug-in do různých projektů vytvářených v UE4 od verze 4.15. Plug-in pro generování mapy byl použit v projektu pro testování chování uživatelů při procházení bludištěm a byl testován studenty předmětu Modelování v grafických aplikacích. Součástí aplikace, která je obsažena v elektronické příloze této práce, je scéna s bludištěm, určená pro odzkoušení generátoru map.

4.4 Mapy ve virtuální realitě

Během vytváření pluginu pro generování map virtuálního světa se vyskytl problém, kdy nebylo možné použít klasické uživatelské rozhraní pro zobrazování mapy ve virtuální realitě, protože wigety se nezobrazovali korektně a dále nebyla možná standardní interakce mezi uživatelem a UI vzhledem k absenci klávesnice. Tyto problémy vedly k vytvoření speciálního 3D widgetu, který se zobrazuje jako součást scény.

Během implementaci byla vytvořena nová třída *VRMapBP* pro generátor mapy, jejíž instance je vkládána do scény pro generování mapy. Třída je implementována obdobně jako třída pro generování mapy zobrazované na monitor uživatele. Rozdíl mezi generátory je v zobrazování uživatelského rozhraní. Mapa je zobrazována jako 3D widget ve scéně. Pro interakci uživatele s rozhraním mapy byla použita komponenta „Widget Interactive“, která umožňuje zachytávat kolize charakteru hráče s 3D widgetem, na základě čehož byly vytvořeny metody pro obsloužení jednotlivých tlačítek pro práci s mapou (Změna měřítka, posunování v mapě). Obrázek vytvořený použitou Scene Capture kamerou je pak použit v uživatelském rozhraní, které je zobrazováno přímo ve scéně jako 3D widget. Příklad vytvořeného widgetu s generovanou mapou ve virtuální realitě je zobrazen na obrázku 24.



Obrázek 24: Příklad možného řešení zobrazování mapy ve virtuální realitě.

4.5 Nároky na výpočet při renderování mapy

Vzhledem k tomu, že snímání scény více kamerami, způsobí vyšší nároky na výpočetní výkon bylo potřeba zjistit do jaké míry se změní počet vykreslených snímků za vteřinu (FPS) a jakým způsobem lze tomuto jevu předejít. Generátor map, je implementovaný tak, aby se jeho kamera, která snímá povrch scény aktivovala až při zobrazení mapy na obrazovku uživatele, proto bylo testováno o kolik se sníží počet snímků při zapnutí mapy. V nastavení kamery lze dále zakázat vykreslování některých efektů a omezit tak počet shader programů, potřebných pro zobrazení snímku mapy. Výsledky různých nastavení kamery a jejich vliv na rychlost výpočtu jsou zobrazeny v tabulce1. Testování bylo prováděno v základní scéně UE4 na počítači s touto konfigurací:

CPU - Intel Core i5 4690K QuadCore (4,5 GHz)

RAM - DDR3 32 GB, 2400MHz

GPU - GeForce GTX970-OC

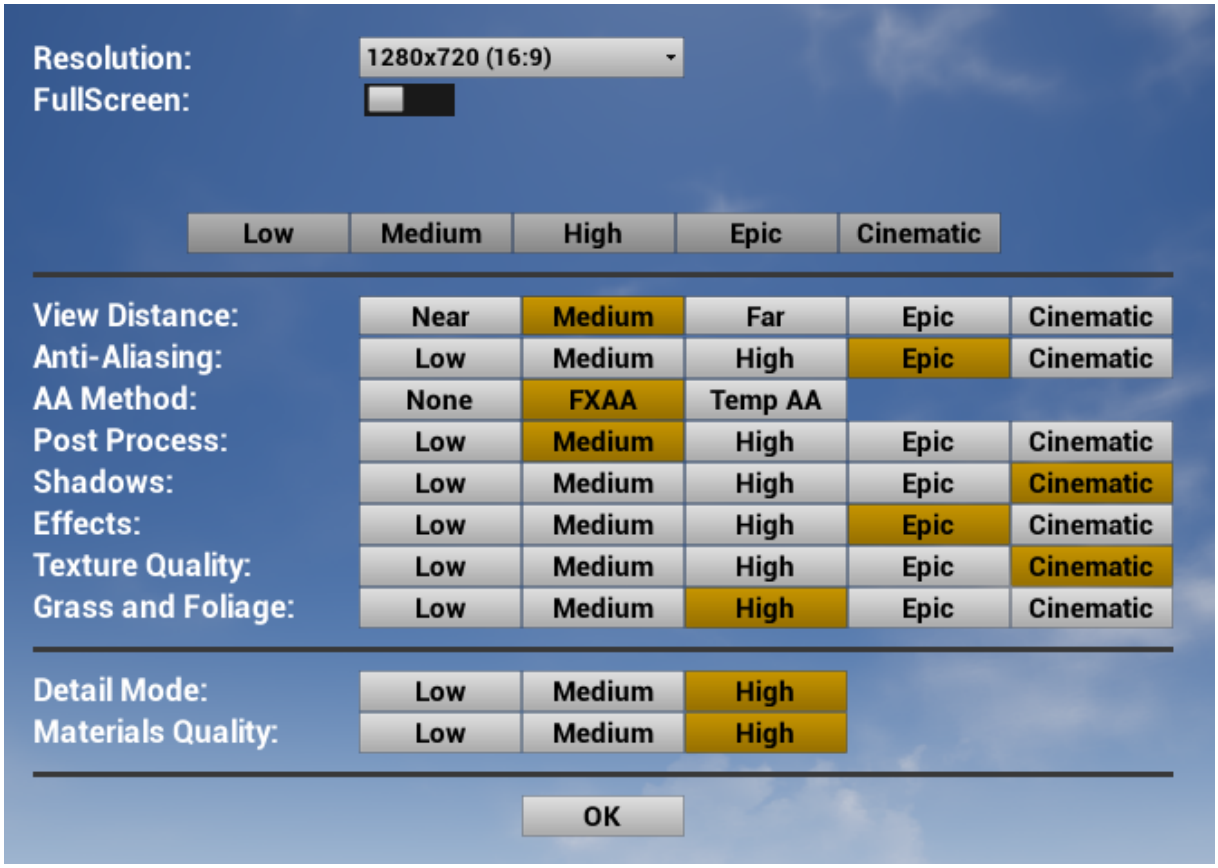
Tabulka 1: Vliv renderování mapy na rychlost vykreslování celé scény

Počet snímků za sekundu (FPS)	Čas renderování jednoho snímku [ms]	Popis
720	1,38	Renderovaná scéna bez zobrazené mapy.
480	2,08	Mapa zobrazena se standardním nastavením.
520	1,92	Vypnuté vyhlazování hran u zobrazované mapy.
580	1,72	Vypnuto renderování všech post-process efektů, částicových systémů, transparentních a odrazivých materiálů u zobrazované mapy.

5 Menu pro nastavení kvality a grafických možností

Během vytváření a testování projektu pro vizualizaci provozu jaderné elektrárny, se objevily problémy s rychlostí výpočtu zobrazované scény, proto bylo potřeba nároky na výpočet nějakým způsobem redukovat přímo za běhu programu, což vedlo k vytvoření menu pro nastavování grafických možností. Pro realizaci byly použity techniky pro tvorbu uživatelského rozhraní integrované přímo v UE4. Pro nastavování úrovně detailů grafických efektů a metod pak byly vytvořeny funkce, které vykonávají změny referenčních proměnných přímo v konzoli UE4, která běží na pozadí aplikace.

Tato kapitola je zaměřena na tvorbu uživatelského rozhraní v UE4 - UI editoru. Vytvořené uživatelské rozhraní pak bylo použito pro možnosti nastavení kvality grafických efektů a metod pomocí příkazů pro konzoli UE4, jejichž popis je taktéž zahrnut v této kapitole. Závěr kapitoly je věnován popisu praktického řešení ukládání a načítání nastavení provedených uživatelem do konfiguračního souboru.



Obrázek 25: Vytvořené menu pro nastavování grafických možností.

5.1 Unreal Motion Graphics UI Designer

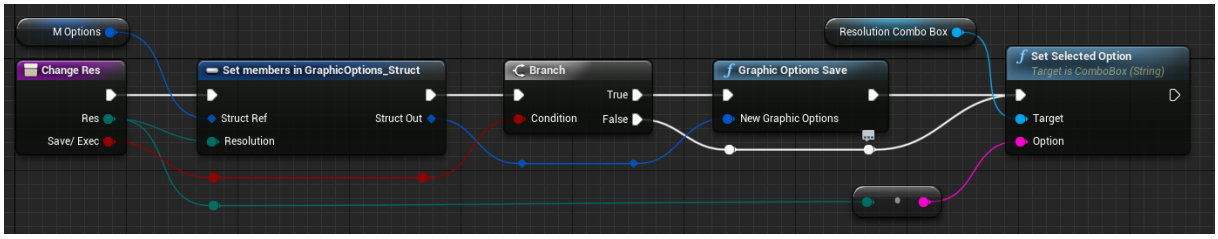
Pro tvorbu uživatelského rozhraní v UE4 slouží tzv. Unreal Motion Graphics User Interface Designer, zkráceně UMG UI Designer. Pomocí tohoto nástroje lze v UE4 vytvářet aktivní aplikace tzv. widgety, ve kterých pomocí standardních prvků pro uživatelské rozhraní jako jsou tlačítka posuvníky apod. lze zobrazovat informace ze zobrazované scény, popřípadě lze widgety použít pro interakci uživatele s aplikací. Výhodou těchto widgetů je, že je lze instancovat přímo ve scéně a jednoduše jim tak předat požadované hodnoty jako parametry, například zobrazovat polohu ostatních uživatelů apod. Na tomto principu bylo postaveno i zobrazování map virtuálního světa popisované v předchozí kapitole.

Při návrhu uživatelského rozhraní menu bylo vycházeno z již zaběhlých standardů používaných u většiny 3D grafických aplikacích a počítačových her. Vzhledem k tomu, že menu mělo být navrženo tak, aby se dalo použít ve více projektech a bylo do jisté míry univerzální, byly při tvorbě UI použity elementy se základním grafickým designem, který nabízí UE4. Jednotlivé možnosti nastavení taktéž vychází z možností nastavení editoru UE4.

Vytvořené uživatelské rozhraní menu je zobrazeno na obrázku 25. Jednotlivé tlačítka menu provádí změny úrovní nastavení daných grafických efektů v reálném čase, tak aby bylo možné změny pozorovat přímo na zobrazované scéně na pozadí menu.

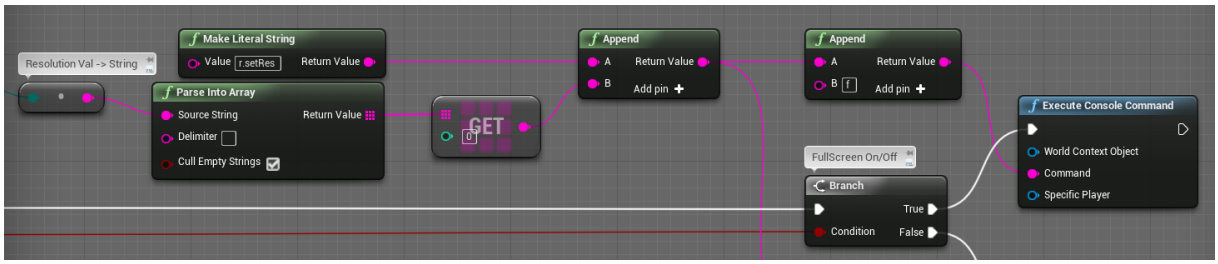
5.2 Interakce s uživatelem a zpracování změny nastavení

Interakce uživatele s jednotlivými elementy UI jsou zachytávány a pomocí implementovaných metod obslouženy. Metody provádějící změnu nastavení jsou implementovány nezávisle na UI, tak že rozhraní pouze provádí změny ve struktuře obsahující informace o nastavení a funkce implementované v UI pouze provádí čtení a zápis hodnot této struktury. V případě změny některé hodnoty ve struktuře se tato změna uloží do konfiguračního souboru a současně se provádí zápis odpovídajícího příkazu do konzole UE4, který provede změnu úrovně grafického nastavení na základě předávaného parametru. Jednotlivé příkazy pro konzoli UE4 použité při implementaci jsou popisovány níže.



Obrázek 26: Obsloužení změny hodnoty rozlišení v uživatelském rozhraní.

Příklad funkce volané po změně rozlišení je zobrazen na obrázku 26. Funkce provede změnu hodnoty parametru *Resolution* ve struktuře *Options*. Následně je volána metoda *Graphic Options Save*, která provede uložení do souboru. Součástí vykonávání metody *Save* je volání metody *Execute*, která na základě struktury s informacemi o nastavení provede odpovídající příkazy do konzole, což vyvolá změnu daného nastavení. Příklad vykonání změny rozlišení je zobrazen na obrázku 27. Podobně jsou prováděny změny i ostatních možností grafického nastavení. Konkrétně při nastavování rozlišení je dále v UI pro menu volána funkce *Init*, tak aby se jednotlivé elementy UI překreslili v nově nastaveném rozlišení.



Obrázek 27: Provedení změny rozlišení na požadovanou hodnotu.

5.3 Sada příkazů pro konzoli UE4

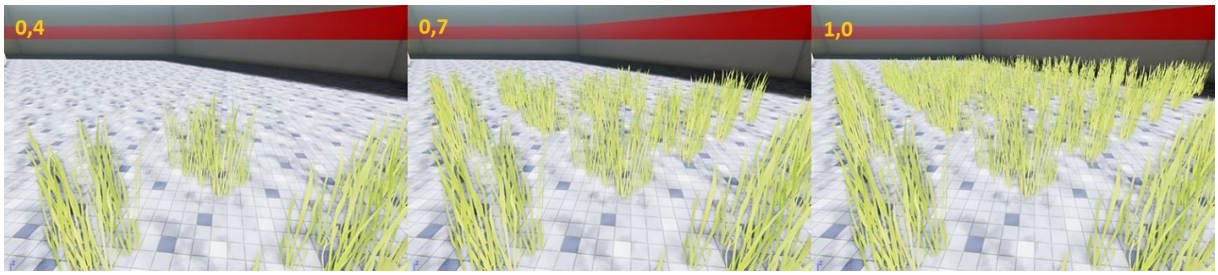
Některá nastavení, popřípadě ladící nástroje lze za běhu aplikace spouštět a nastavovat pomocí příkazů zadávaných do UE konzole. Nastavování úrovní kvality jednotlivých grafických efektů a metod lze provádět pomocí speciálních příkazů. Pro zasílání příkazů do konzole slouží v UE4 funkce *Exec Console Command*, které se jako parametr předá text příkazu, který se má vykonat. Za běhu aplikace lze konzoli vyvolat standardním tlačítkem „**;**“. (Tlačítko lze změnit v nastavení projektu, popřípadě lze konzoli úplně zakázat.) Seznam všech základních příkazů pro UE konzoli je k dispozici v dokumentu na stránkách Epic Games [19]. Příkazy které byly použity při implementaci jsou popsány v následujících částech.

5.3.1 Nastavení rozlišení

Jednou ze základních a nejpoužívanější možnosti při modifikaci grafického nastavení v počítačových hrách patří změna rozlišení okna aplikace. Pro změnu rozlišení slouží v UE příkaz *r.SetRes*, pomocí kterého lze rozlišení okna nastavit na téměř libovolnou hodnotu i v libovolném poměru. Za příkazem následuje parametr, určující hodnotu rozlišení například „800x600w“ popřípadě „800x600f“, kde suffixy w a f určují zda aplikace poběží v okně, nebo přes celou obrazovku. Při tvorbě menu pro nastavování grafických možností byl vytvořen výčtový typ, který volbu rozlišení omezuje na standardně používané hodnoty dle statistik služby Steam [20].

5.3.2 Vzdálenost vykreslování

Pro nastavení vzdálenosti vykreslování (tzv.: View Distance) slouží příkaz *r.ViewDistanceScale*, jehož parametrem je hodnota z intervalu $< 0.0; 1.0 >$. Hodnota parametru udává vzdálenost od pozorovatele, pro kterou budou objekty ve scéně renderovány, vzhledem k maximální vykreslovací vzdálenosti nastavené při implementaci aplikace. Objekty vzdálené od kamery více něž udává nastavená hodnota jsou pro renderovaní ze scény ořezány. Výchozí nastavení je rovno 0 což znamená, že vzdálenost vykreslování není nastavena a do renderování budou zahrnuty všechny objekty scény, popřípadě bude použita základní hodnota nastavená při sestavování scény. Maximální hodnota (1.0) pak nastavuje vzdálenost vykreslování na maximální hodnotu, které odpovídá 1000 UE jednotek. Změna vzdálenosti vykreslování je zobrazena na obrázku 28



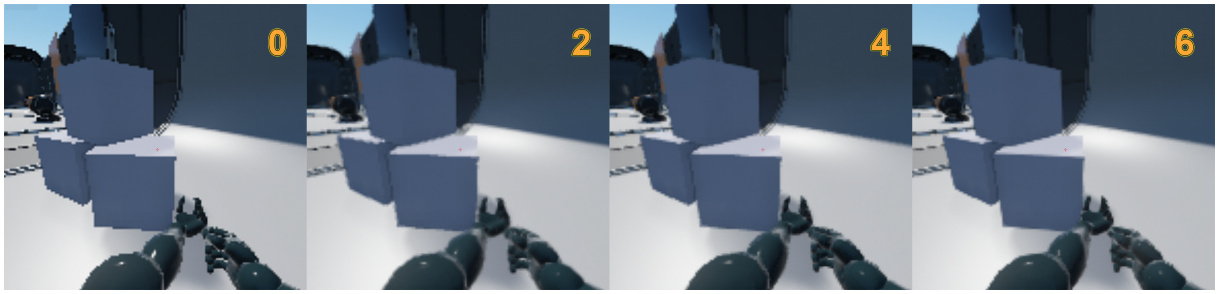
Obrázek 28: Nastavení vzdálenosti vykreslování a dopad na renderovanou scénu. [21]

5.3.3 Vyhlazování hran (Anti-Aliasing)

Nastavením úrovně kvality vyhlazování hran, lze v UE aplikaci provést pomocí příkazu *r.Post Process AA Quality*, který jako parametr přijímá celočíselnou hodnotu z intervalu $< 0; 6 >$, kde 0 představuje úplné potlačení vyhlazovací metody. Potlačení metody pro vyhlazování hran může do velké míry urychlit vykreslování scény, ovšem na úkor kvality zobrazovaných objektů (viz obrázek 29).

Anti-Aliasing lze ovlivnit i pomocí zvolené metody pro vyhlazování hran. UE4 nabízí několik metod pro vyhlazování, mezi které patří například FXAA, nebo Temporal AA (TXAA). Změnu používané metody za běhu aplikace UE4 neumožňoval pomocí příkazu pro konzoli do verze 4.13 a bylo nutné implementovat vlastní funkci, která by tento nedostatek kompenzovala. Po vydání verze UE 4.13 se v seznamu příkazů objevil příkaz *r.Default Feature. AntiAliasing*, který umožňuje přepínání jednotlivých metod vyhlazování hran přímo v konzoli UE. Proto bylo i vytvářené menu aktualizováno o použití tohoto příkazu, namísto dříve implementované metody, která potřebovala referenci na kameru snímající scénu.

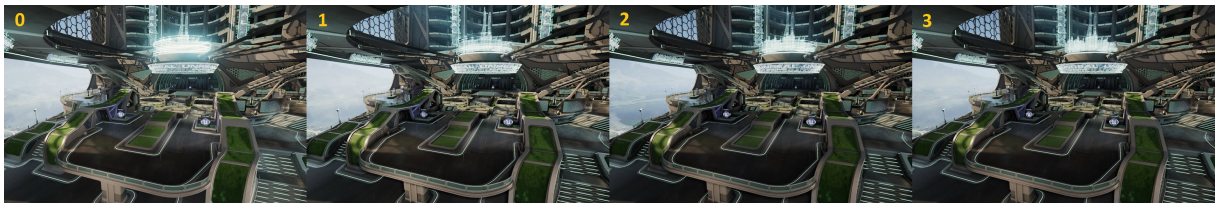
V případě FXAA metody pro vyhlazování hran vývojáři UE4 [21] doporučují použít nejvyšší nastavení úrovně na hodnotu 2 (v menu této hodnotě odpovídá úroveň „Medium“), vyšší úroveň již výsledný obraz neovlivní, pouze je metoda více náročná na výpočetní výkon. Příklad vykreslené scény s jinými hodnotami úrovně kvality vyhlazování hran je vidět na obrázku 29.



Obrázek 29: Použití různých úrovní kvality vyhlazování (TXAA). [21]

5.3.4 Zpracovávání výsledného obrazu (Post-Processing)

Dalším způsobem, kterým lze ovlivnit rychlost vykreslování scény je nastavením úrovně výsledného zpracování obrazu resp. Post-Processingu. K tomu slouží příkaz *sg.PostProcessQuality*, jehož parametrem je celočíselná hodnota z intervalu $< 0; 3 >$, kde jednotlivé hodnoty představují úrovně „Low“, „Medium“, „High“ a „Epic“. Od verze Unreal Engine 4.15 vydané v březnu roku 2017 lze použít i hodnotu 4 pro nastavení na úrovně „Cinematic“. Implementované menu bylo na základě těchto nových možností rozšířeno o tuto další možnou úroveň nastavení.



Obrázek 30: Úrovně kvality vykreslování Post-Process efektů. [21]

Změnou hodnoty použitím *sg.PostProcessQuality* se vyvolá sekvence příkazů, které ovlivní kvalitu jednotlivých post-process efektů, aplikovaných na renderovaný obraz. Jednotlivé příkazy provedené na základě změny hodnoty úrovně kvality zpracování výsledného obrazu jsou uvedeny v následující tabulce (Tabulka 2. Výsledný dopad změny nastavení kvality vykreslování post-process efektů je zobrazen na obrázku 30.

Tabulka 2: Proměnné a hodnoty nastavení po provedení příkazu *sg.PostProcessQuality*.

PostProcessQuality	LOW	MEDIUM	HIGH	EPIC
MotionBlurQuality	0	3	3	4
BlurGBuffer	0	1	-1	-1
AmbientOcclusionLevels	0	1	2	3
AmbientOcclusionRadiusScale	1.7	1.7	1.5	1.0
DepthOfFieldQuality	0	1	2	2
RenderTargetPoolMin	300	350	400	400
LensFlareQuality	0	0	2	2
SceneColorFringeQuality	0	0	1	1
EyeAdaptationQuality	0	0	2	2
BloomQuality	4	4	5	5
FastBlurThreshold	0	2	3	7
Upscale.Quality	1	2	2	3
Tonemapper.GrainQuantization	0	0	1	1

5.3.5 Kvalita zobrazování stínů

Kvalitu stínů lze nastavit obdobně jako kvalitu Post-Process efektů, pomocí příkazu *sg.Shadow Quality* s parametrem odpovídajícím nastavované hodnotě z intervalu $< 0; 4 >$. Změna hodnoty úrovně kvality stínů vyvolá sekvenci příkazů, které jsou popsány v následující tabulce 3. Jednotlivé příkazy pak ovlivňují nastavení prvků souvisejících s vykreslováním stínů. Dopad nastavení na renderovanou scénu je zobrazen na obrázku 31.

Tabulka 3: Proměnné a hodnoty nastavení po provedení příkazu *sg.ShadowQuality*.

ShadowQuality	LOW	MEDIUM	HIGH	EPIC
LightFunctionQuality	0	1	1	1
ShadowQuality	0	2	5	5
Shadow.CSM.MaxCascades	1	1	2	4
Shadow.MaxResolution	512	1024	1024	1024
Shadow.RadiusThreshold	0.06	0.05	0.04	0.03
Shadow.DistanceScale	0.6	0.7	0.85	1.0
Shadow.CSM.TransitionScale	0	0.25	0.8	1.0

Obrázek 31: Úrovně kvality vykreslování stínů. [21]

5.3.6 Kvalita textur

Nastavení kvality textur lze provést pomocí příkazu *sg.TextureQuality*, jehož parametry jsou stejné jako pro nastavení kvality stínů či post-process efektů. Obdobně jako v předchozích případech se po změně hodnoty úrovně kvality textur provede sekvence příkazů s danými parametry, které jsou popsány v následující tabulce 4. Výsledek nastavení závisí i na renderované scéně a použitém hardwaru. Pokud je ve scéně použito minimum textur a grafická karta má dostatek volné paměti nebude znatelný žádný rozdíl v jednotlivých úrovních nastavení.

Tabulka 4: Proměnné a hodnoty nastavení po provedení příkazu *sg.TextureQuality*.

TextureQuality	LOW	MEDIUM	HIGH	EPIC
Streaming.MipBias	2.5	1	0	0
MaxAnisotropy	0	2	4	8
Streaming.PoolSize	200	400	700	1000

5.3.7 Kvalita efektů

Nastavení kvality efektů se týká především použitého materiálu u jednotlivých objektů. Nastavení se provádí obdobně jako u nastavení stínů, a to příkazem *sg.EffectsQuality* zadaným do UE konzole. Hodnoty parametru jsou opět z intervalu $< 0; 4 >$ a změna úrovně vyvolá stejně jako v předchozích případech sekvenci příkazů ovlivňující jednotlivá nastavení související s kvalitou vykreslování efektů. (Viz tabulka 5)

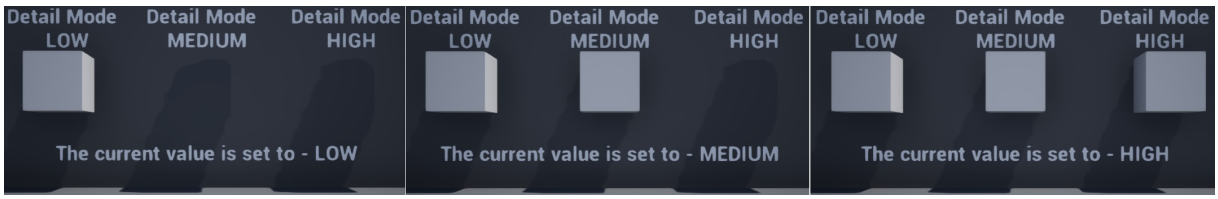
Tabulka 5: Proměnné a hodnoty nastavení po provedení příkazu *sg.EffectsQuality*.

ShadowQuality	LOW	MEDIUM	HIGH	EPIC
TranslucencyLightingVolumeDim	24	32	48	64
RefractionQuality	0	0	2	2
SSR	0	0	0	1
SceneColorFormat	3	3	3	4
DetailMode	0	1	1	2
TranslucencyVolumeBlur	0	0	1	1
MaterialQualityLevel	0	1	1	1

5.3.8 Úroveň detailů

Umožňuje nastavit úroveň detailů renderování jednotlivých aktorů ve scéně, kde každému aktéru lze nastavit možnost, ve které úrovni detailů bude renderován. Na základě tohoto nastavení lze ze scény vypustit některé efekty, dokonce i objekty, jako jsou Decals projektory, statické modely nebo částicové systémy. Nastavení úrovně detailů lze pak provést pomocí příkazu *r.DetailMode*, s parametry 1, 2 nebo 3.

Dopad změny nastavení úrovně detailů na renderovanou scénu ve které jsou objekty s různými hodnotami Detail Mode je zobrazen na obrázku 32.



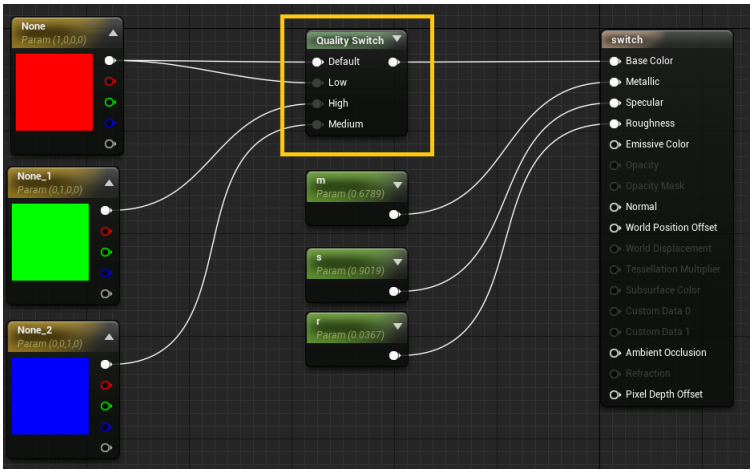
Obrázek 32: Renderování scény s různým nastavením Detail Mode.

5.3.9 Úroveň kvality materiálů

Umožňuje měnit kvalitu grafických efektů přímo na úrovni materiálů. Vytvořené materiály musí být pro tuto možnost přizpůsobeny použitím funkce „quality switch“, která je součástí materiál editoru v UE. Tato funkce umožňuje přepnout například použité textury, nebo aplikované masky a tím ovlivnit zásadní vlastnosti materiálu například intenzitu odrazů. Nevýhodou je, že každým použitím přepínače se zvýší počet shaderů potřebných k kompilaci materiálu. Shadery se kompilují zvlášť pro každou možnost přepínače.

Obrázek 33: Dopad změny úrovně kvality materiálu.

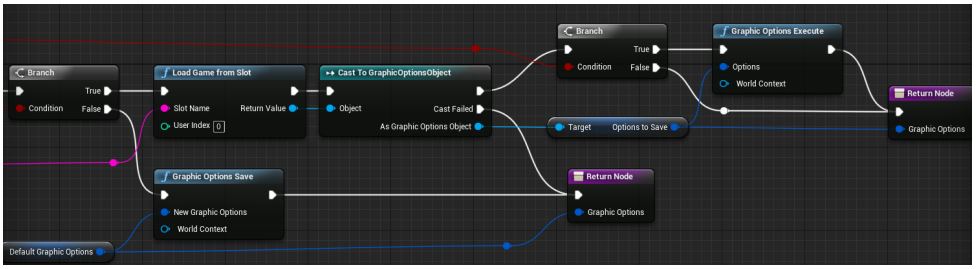
Za běhu aplikace pak lze nastavení kvality materiálů měnit pomocí příkazu *r.MaterialQualityLevel* kde parametrem jsou hodnoty 0 pro „Low“ kvalitu a 1 pro „High“ kvalitu. Od verze UE 4.8 byla přidána možnost medium a funkce přepínače v editoru materiálu, tak byla rozšířena o další úroveň. Použití přepínače v materiálu je zobrazeno na obrázku 33, dopad změny nastavení úrovně kvality materiálu na renderování objektu s daným materiálem je pak zobrazeno na obrázku 33.



Obrázek 34: Použití přepínače při vytváření materiálu.

5.4 Ukládání hodnot nastavení

Jednotlivé hodnoty nastavení úrovně grafických možností jsou v UE ukládány do globální struktury. Tato struktura byla vytvořena v tzv. „GameInstance“, kde k ní mají přístup všechny metody a třídy vytvořené v rámci aplikace. Toto řešení ovšem přineslo problémy s přenositelností zdrojových kódu pro menu mezi různými projekty. Proto bylo řešení pomocí „GameInstance“ a globální struktury nahrazeno třídou, která obsahuje tuto strukturu a metody sloužící k práci s touto strukturou.

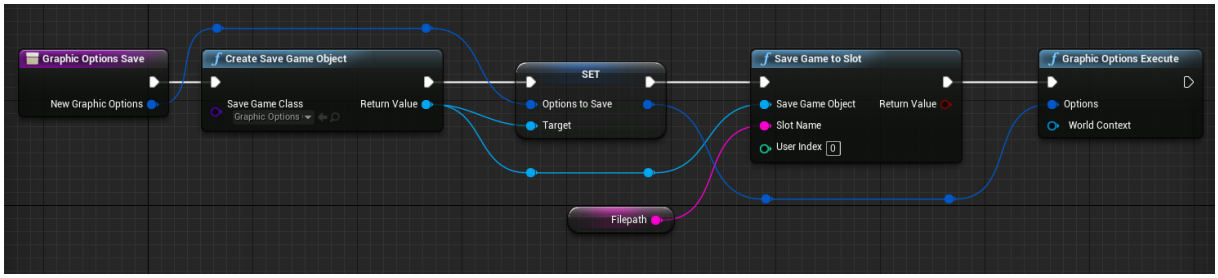


Obrázek 35: Načtení uložených hodnot nastavení ze souboru.

Jednotlivé metody jsou pak nastaveny jako globální a lze je volat ze všech tříd použitých v aplikaci. Pro naplnění struktury je pak potřeba zavolat funkci *Load* při spouštění aplikace, která provede načtení hodnot ze souboru a jejich uložení do struktury, se kterou se bude dále pracovat v rámci aplikace. Funkce *Load* současně volá metodu *Execute*, která provede nastavení grafických možností na požadované hodnoty, které byly uloženy v souboru. Metoda Execute již byla popsána v části 5.2. V případě, že není nalezen soubor s nastavením, je vytvořen nový soubor s hodnotami nastavení grafických možností na nejnižší úroveň. Příklad implementované

metody *Graphic Options Load* pro načtení hodnot ze souboru je zobrazen na obrázku 35.

Pokud uživatel pomocí uživatelského rozhraní změní některé z nastavení, je v rámci vykonávání obslužných funkcí (viz část 5.2) volána metoda *Graphic Options Save*, která provede uložení aktuálních hodnot ve struktuře a současně volá i metodu *Execute*.

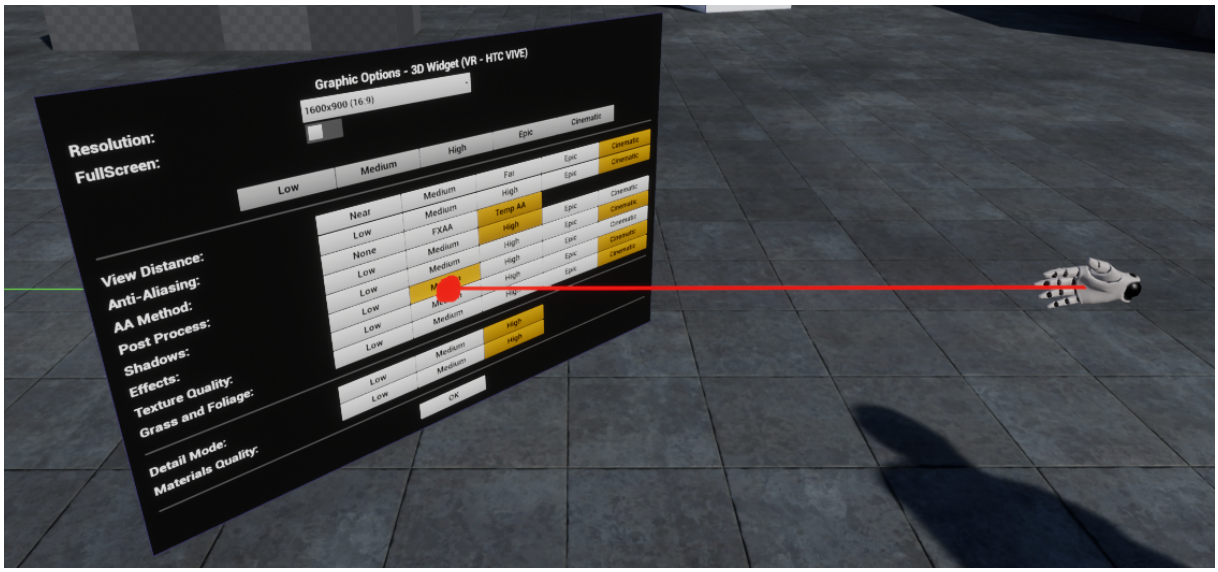


Obrázek 36: Uložení hodnot nastavení grafických možností do souboru.

Ukládání hodnot do konfiguračního souboru je prováděno pomocí metody *SaveGame* implementované vývojáři UE4, pro ukládání aktuálního stavu aplikace. Tato metoda provede serializaci předem definovaných datových struktur a uloží je do souboru (.sav) na disk. V případě implementovaného menu se provede serializace pouze struktury obsahující aktuální hodnoty úrovní nastavení. Načítání je pak provedeno obdobně pomocí metody *LoadGame*, která parsuje hodnoty z uloženého souboru a naplní jimi požadovanou datovou strukturu. Příklad implementované metody pro ukládání aktuálního stavu nastavení grafických možností je zobrazen na obrázku 36.

5.5 Menu pro nastavení ve virtuální realitě

Podobně jako u zobrazování mapy virtuálního světa, vyskytli se i u zobrazování widgetu menu pro nastavení grafických možností problémy s korektním zobrazením widgetu na brýlích pro virtuální realitu. Uživatelské rozhraní se opět zobrazovalo mimo zorné pole uživatele a nebylo možné provádět interakce s jednotlivými tlačítky nastavení kvůli absenci klávesnice a myši. Řešením tohoto problému bylo použití 3D widgetu zobrazovaného jako součást scény. Pro uživatelsky příjemné používání byl 3D widget vytvořen tak aby se zobrazoval přímo na „virtuální ruce“ uživatele. Konkrétně bylo využito ovladače pro levou ruku, na jehož pozici se menu zobrazuje. Použití vytvořeného 3D widgetu ve virtuální realitě je zobrazeno na obrázeku 37

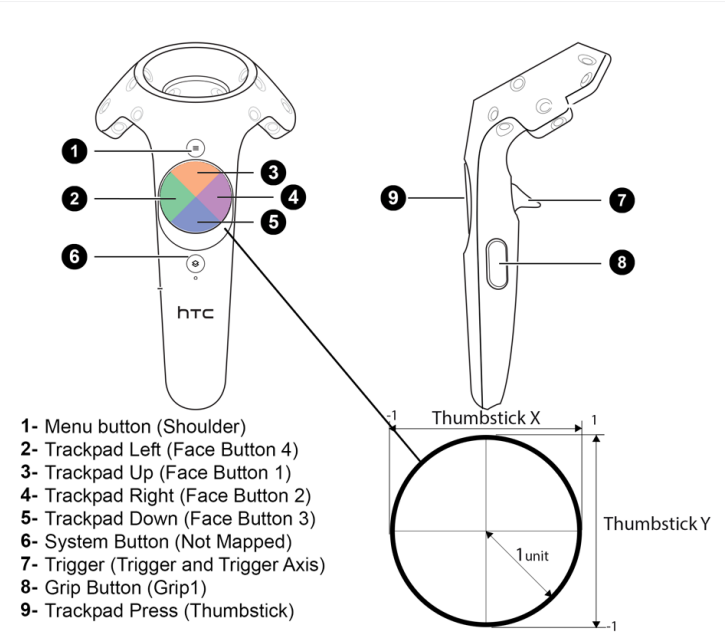


Obrázek 37: Zobrazení menu pro nastavení grafických možností ve virtuální realitě.

V rámci implementace byla vytvořena třída *VR 3D widgets*, jejíž instance je vkládána do scény jako aktér a je jí předávána reference na třídu *VRPawn*, která ve scéně reprezentuje uživatele ve virtuální realitě. Ve třídě *VR 3D widgets* pak byly vytvořeny metody pro obsluhování vstupů od uživatele a funkce pro zobrazení widgetu na ruce uživatele. Třída byla vytvořena tak, aby se dala použít pro zobrazování více widgetů současně. Reference na jednotlivá UI jsou třídě předávány jako parametry.

5.5.1 Ovládání a chování widgetu

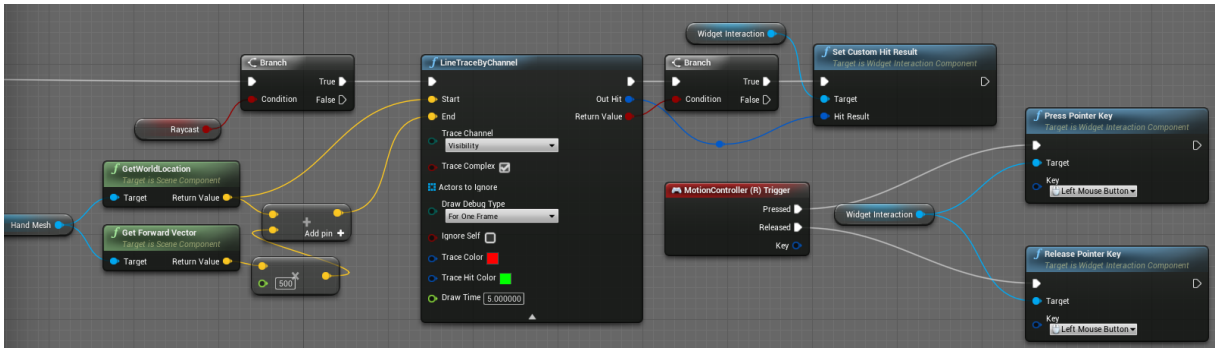
Během implementace bylo potřeba zmapovat jednotlivé funkční prvky ovladačů tak aby se daly využít pro interakci s 3D widgetem. Označení a význam jednotlivých funkčních tlačítek na ovladačích pro virtuální realitu (HTC - VIVE) je zobrazeno na následujícím obrázku 38.



Obrázek 38: Namapování tlačítek ovladačů pro virtuální realitu.

Pro zobrazení wigetu na ruce uživatele bylo použito tlačítko na ovladači (ovladač pro levou ruku) určené pro zobrazování menu (tlačítko *Shoulder* - viz obrázek 38). Po stisknutí je na místě levého ovladače ve virtuální realitě zobrazeno první UI z pole widgetů. Pro přepínání mezi zobrazovanými widgety pak byly využity tlačítka *Trackpad Left* a *Trackpad Right* na levém ovladači (viz obrázek 38), které provádí přiřazení UI do 3D widgetu na základě pozice aktuálního UI v poli widgetů.

Pro interakci uživatele s jednotlivými prvky UI, bylo využito ovladače pro pravou ruku, kde pomocí metody sledování paprsku je získávána reference na konkrétní tlačítko UI a pomocí přepínače na ovladači pravé ruky (tlačítko Trigger) je volána funkce daného tlačítka uživatelského rozhraní. Implementovaná metoda ve třídě *VR 3D widgets* pro získání reference na dané tlačítko a následné volání funkce pro jeho obsloužení je zobrazena na obrázku 39. Sledování paprsku ve scéně je patrné i na obrázku 37.



Obrázek 39: Funkce pro získání reference na tlačítka z UI na které uživatel ukazuje ovladačem.

5.6 Vliv nastavení na výsledek renderování

Pro demonstraci účinnosti změny nastavení grafických možností na rychlost vykreslování vizualizované scény bylo provedeno testování jednotlivých nastavení. Protože rychlost renderování je závislá na rozložení scény, použitých efektech a složitosti modelů bylo testování menu provedeno na několika scénách, a to na základní scéně UE4, scéně „Infinity Blade“ a venkovní scéně projektu Virtuální jaderné elektrárny. U jednotlivých scén byl vyhodnocován počet renderovaných snímků za sekundu, v závislosti na zvoleném nastavení v menu. Jednotlivé hodnoty pak byly zapsány do tabulky (viz tabulka 6). Testování bylo prováděno na počítači s touto konfigurací:

CPU - Intel Core i5 4690K QuadCore (4,5 GHz)

RAM - DDR3 32 GB, 2400MHz

GPU - GeForce GTX970-OC

Tabulka 6: Rychlosti vykreslování scény v závislosti na zvoleném nastavení.

Úroveň nastavení	Základní scéna UE4 FPS	Infinity Blade Demo FPS	Jaderná elektrárna FPS
LOW	690	230	290
MEDIUM	390	170	230
HIGH	325	150	200
EPIC	300	135	150
CINEMATIC	290	130	120
Pozn.: Rozlišení okna bylo při testování nastaveno na 1600x900px. Hodnota FPS udává průměr za 1 minutu procházení scény.			

6 Možnosti ovládání pohyblivých objektů ve scéně

Během vytváření projektu pro vizualizaci provozu jaderné elektrárny bylo potřeba implementovat funkce pro pohybování s objekty ve scéně. Například pohyb zavážejícího stroje, přeprava paliva, manipulace s kontejnery ve skladu apod.

6.1 Geometrické transformace ve 3D

Pro změnu polohy objektu v zobrazované 3D scéně, změnu velikosti objektu nebo jeho natočení slouží v počítačové grafice geometrické transformace. V počítačové grafice se využívá tzv. projektivního prostoru a projektivních transformací v homogenní souřadné soustavě, neboť vzhledem k afinním transformacím, počítaným v kartézském souřadném systému, umožňují projektivní transformace realizovat rovnoběžné i středové promítání, kdežto pomocí afinních transformací lze realizovat promítání pouze rovnoběžné. Použití projektivního prostoru dále potlačuje výskyt nevlastních bodů (body v nekonečnu) a transformace v homogenní souřadné soustavě lze jednoduše skládat jako násobení matic.

Matice transformací v homogenní souřadné soustavě používané v počítačové grafice jdou uvedeny níže. V UE4 a prakticky ve všech grafických systémech jsou polohy bodů popřípadě transformace popisovány pomocí afinních souřadnic, ty jsou ovšem pro další zpracování převáděny do prostoru homogenního. Popis převodu a hlubší teorie o souřadných systémech viz [22].

- Translace (posunutí):

$$\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotace kolem osy *x, y, z*:

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\Theta & -sin\Theta & 0 \\ 0 & sin\Theta & cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{R}_y = \begin{bmatrix} cos\Theta & 0 & sin\Theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\Theta & 0 & cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}_z = \begin{bmatrix} cos\Theta & -sin\Theta & 0 & 0 \\ sin\Theta & cos\Theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Změna měřítka:

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6.2 Transformace objektu v UE4

Unreal Engine 4 nabízí několik funkcí pomocí kterých lze provádět základní geometrické transformace s objekty. Tyto funkce pak aplikují danou transformaci na všechny vrcholy tvořící objekt ve scéně. Součástí UE4 je několik typů funkcí pro provádění transformací objektu.

Transformační funkce s prefixem „Set“ slouží v EU4 pro nastavování nové hodnoty jednotlivých transformací daného objektu. Vstupem těchto funkcí jsou vektory, jejichž hodnoty představují novou hodnotu odpovídající transformace, popřípadě novou pozici nebo na-rotování objektu. Tyto hodnoty jsou funkcím předávány jako parametry společně s referencí na objekt, pro který má být transformace provedena. UE4 obsahuje následující sadu funkcí pro nastavování nových transformací ve světovém souřadném systému (World Space) a pro nastavení transformací relativních. Ty jsou vztažené vždy k nadřazenému objektu. Například je li ve scéně umístěn aktor, který je vlastníkem nějakého objektu je možné pomocí funkcí *Set Relative* nastavit pozici objektu vzhledem k souřadnému systému aktéra, kde pozice aktéra ve scéně se nijak nezmění. Pro nastavování nových transformací objektu jsou v UE4 k dispozici tyto funkce:

Set World/Relative Location - Nastaví novou pozici danému objektu.
Set World/Relative Rotation - Nastaví novou hodnotu narotování
Set World/Relative Scale 3D - Umožňuje nastavit změnu měřítka.
Set World/Relative Transform - Definuje danému objektu novou hodnotu všech transformací. (pozici, na-rotování, zvětšení)

Další funkce pro změnu transformací objektu v UE4 jsou funkce provádějící přičítání nové hodnoty transformace k aktuální hodnotě. Obdobně jako u funkcí s prefixem Set lze změnu transformací provádět vzhledem ke světovému souřadnému systému (World), vzhledem k soustavě vlastníka daného objektu (Relative), nebo přímo k lokálnímu systému daného objektu (Local).

Add World/Local/Relative Location
Add World/Local/Relative Rotation
Add World/Relative Transform

6.2.1 Parametry Teleport a Sweep

U jednotlivých výše popisovaných funkcí pro provádění transformací objektů je možné využít i parametrů *Teleport* a *Sweep*. Tyto parametry jsou reprezentovány booleovskou hodnotou, která definuje jakým způsobem se bude změna transformace objektu ve scéně provádět, popřípadě jak bude změna ovlivňovat ostatní objekty ve scéně a jakým způsobem bude reagovat na kolize s ostatními objekty.

Během provádění změny pozice nebo rotace objektu ve scéně je rovněž u objektu počítána jeho rychlost jakou se po scéně přesunuje. V případě změny pozice je u objektu rychlost jeho pohybu odvozena od původní a nové pozice ve scéně. Pro výpočet využívá UE4 jednoduchého vztahu

$$v = \frac{\triangle P}{t}, \tag{10}$$

kde rychlost v je dána vzdáleností mezi původní a novou pozicí $\triangle P$, za čas potřebný k renderování jednoho snímku t. Tato rychlost se pak projevuje v případě kolizí s ostatními objekty ve scéně, kde na základě rychlosti je počítána i kolize objektů. Dále je rychlost předávána i připojeným objektům k objektu posunovanému. V případě, kdy se jedná o fyzický objekt (je počítána fyzika chování objektu) tato rychlost způsobí rozkmitání připojeného objektu po provedení posunu. Pokud by byl objekt přesunován na velkou vzdálenost, byla by i jeho výsledná rychlost během přesunu obrovská a výsledná odstředivý síla by pak působila na připojený objekt. To by mělo za následek například nerealistické chování. Pro potlačení výpočtu rychlosti a její aplikace na připojené objekty lze nastavit parametr *Teleport* na hodnotu 1. S parametrem *Teleport* pak úzce souvisí parametr *Sweep*, který ovlivňuje výpočty kolizí s okolními objekty během provádění transformace.

Přehled jednotlivých konfigurací nastavení parametrů je znázorněn v následující tabulce 7. Animace chování objektů během provádění transformací na základě nastavení parametrů jsou pak k dispozici v online dokumentaci UE viz [23].

Tabulka 7: Nastavení parametrů Teleport a Sweep. [23]

	Teleport = 0	Teleport = 1
Sweep = 0	Výsledná pozice je specifikována uživatelem. Změna rychlosti je závislá na konečné poloze. Kolizní objekty reagují na implicitní rychlost.	Výsledná pozice je specifikována uživatelem. Rychlost neovlivněna.
Sweep = 1	Výsledná poloha je dána kolidujícím objektem v cestě. Změna rychlosti je závislá na konečné poloze. Kolize se neprovede. Sweep neprovede kolizi, objekt se zastaví před kolidujícím objektem	Výsledná poloha je dána kolidujícím objektem v cestě. Rychlost neovlivněna. Kolize se neprovede. Sweep neprovede kolizi, objekt se zastaví před kolidujícím objektem

6.2.2 Metoda „Move Component To“ a její použití

Pomocí cyklického volání transformačních funkcí je také možné jednoduše implementovat metody pro plynulý pohyb objektu ve scéně. Na tomto principu pracuje i metoda *Move Component To*. Aby pohyb nebyl skokový lze interpolovat výchozí polohu objektu a polohu novou tak, aby se během renderování jednotlivých snímků vizualizované scény měnila pozice po malých krocích, což navozuje dojem plynulého pohybu objektu. Metodě *Move Component To* jsou jako parametr předávány následující hodnoty:

Component - Reference na objekt ve scéně.

Target Relative Location - Vektor nové pozice objektu

Target Relative Rotation - Vektor nového na-rotování objektu.

Ease Out - určuje zda má byt interpolace pohybu lineární nebo má mít logaritmický průběh (konec se bude provádět po menších krocích).

Ease In - určuje zda má byt interpolace pohybu lineární nebo má mít logaritmický průběh (začátek se bude provádět po meších krocích).

Over Time - Hodnota určující po jakou dobu se bude metoda vykonávat (určuje délku časové osy). Na základě času se pak počítá velikost jednotlivých kroků interpolace.

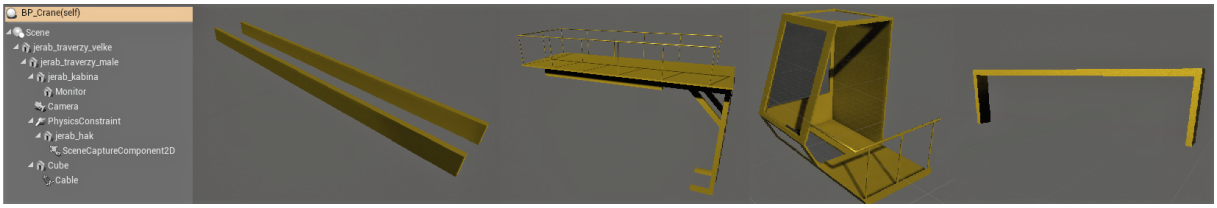
Obrázek 40: Funkce pro ovládání pojezdových vrat ve skladu.

V projektu pro vizualizaci provozu jaderné elektrárny byla metoda *Move Component To* využita pro simulaci pohybu otevírání a zavírání posuvných vrat ve skladu použitého paliva. Implementovaná funkce pro otevírání a zavírání vrat je zobrazena na obrázku 40. Na základě hodnoty proměnné *Door Is Open*, která udává aktuální stav objektu, ze zavolá metoda *Move Component To*, které je předána reference na objekt (Vrata) a vektor udávající novou pozici na kterou se za danou dobu objekt posune.

Funkce je volána ze spouštěče(Triggeru) umístěného ve scéně před vraty. V případě, kdy uživatel vstoupí do jeho oblasti je funkce volána pro otevření zavřených vrat. Opustí-li uživatel prostor spouštěče je zavolána funkce znovu a vrata se začnou zavírat. Umístění objektu a triggeru ve scéně je rovněž zobrazeno na obrázku 40.

6.3 Implementace manipulačního jeřábu ve skladu

V rámci praktické části byl vytvořen model jeřábu pro manipulaci s kontejnery ve skladu použitého paliva jaderné elektrárny. Pro ovládání jeřábu byly implementovány metody využívající funkce pro transformace objektů dostupné v UE4, tak aby bylo možné jeřáb ovládat a používat jej k přemisťování dalších objektů ve scéně.



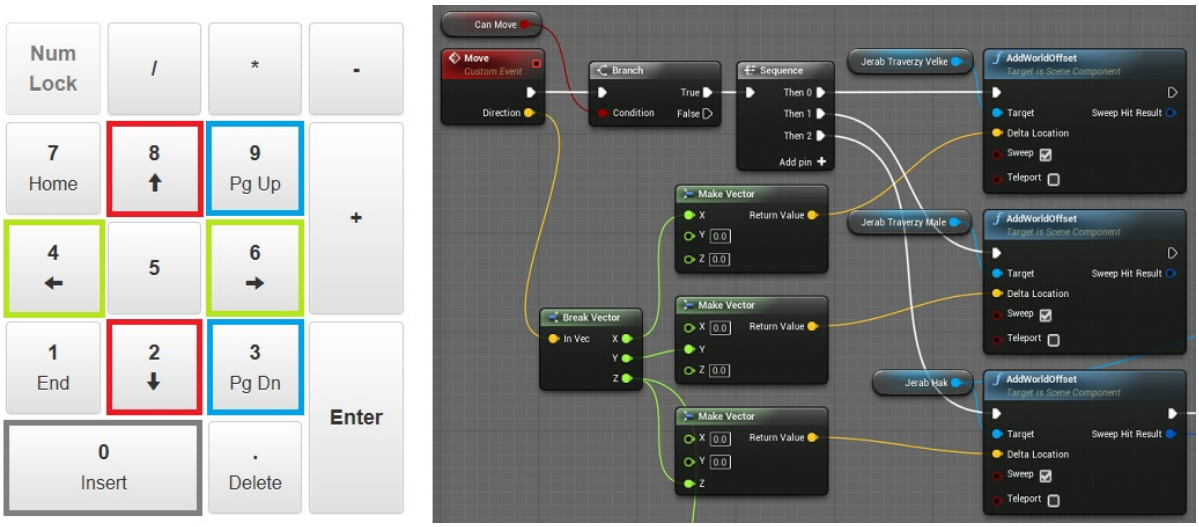
Obrázek 41: Struktura řazení a jednotlivé modely jeřábu.

Model jeřábu se skládá z několika částí, které jsou do UE importovány jako samostatné modely. Do scény je pak vkládán aktor (instance třídy Crane), který je vlastníkem těchto modelů a metod pro pohybování s jednotlivými částmi.

Jednotlivé modely, ze kterých se jeřáb skládá jsou následující: statické kolejnice umístěné na pilířích skladu (nejsou součástí třídy *Crane*). Hlavní traverzy pohybující se dopředu a dozadu. Po hlavních traverzách se pohybuje plošina s kabinou jeřábu, které se pohybují doleva a doprava. Posledním modelem je pak samotný hák jeřábu, který vykonává pohyby nahoru a dolů. Jednotlivé modely jsou hierarchicky uspořádány tak aby pohyb jednoho ovlivnil modely jemu podřízené. Modely jeřábu a jejich hierarchické řazení ve třídě *Crane* je zobrazen na obrázku 41.

Strukturované řazení modelů zaručuje, že pokud se budou pohybovat traverzy jeřábu dopředu a dozadu, budou se současně pohybovat i model plošiny a háku. Tento jev je způsoben tím, že model plošiny je umístěn do souřadného systému modelu traverzí, jehož střed je přemísťován pomocí implementovaných funkcí dopředu a dozadu vzhledem k souřadnému systému komponenty „Scene Component“. Navenek se pak jeví že aktor umístěný do scény má stejné souřadnice i přesto, že jeho komponenty se pohybují.

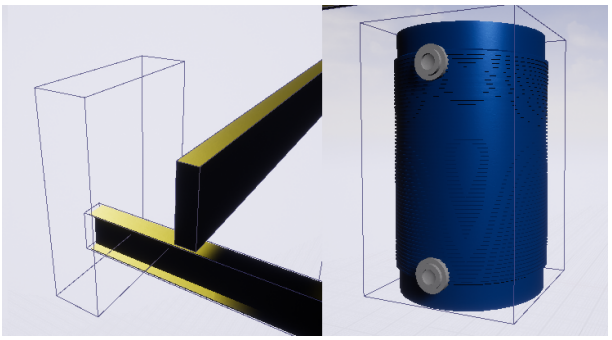
Pro pohybování s jednotlivými modely pak byla implementována metoda *Move*, které je předáván směr pohybu jako parametr. Volání metody *Move* je inicializováno vstupem z klávesnice. Klávesy použité pro ovládání jeřábu metoda *Move* jsou zobrazeny na obrázku 42, kde červená barva odpovídá pohybu v ose x, zelená v ose y, modrá v ose z. Tlačítko numerické nuly bylo použito pro přichycení objektu k jeřábu.



Obrázek 42: Tlačítka klávesnice použitá pro ovládání jeřábu.

6.4 Využití kolizí

Pro ošetření rozsahu pohybů jeřábu a jeho částí byly při implementaci jeřábu použity tzv. kolize a kolizní objekty. Tyto kolizní objekty byly umístěny tak, aby zabránili najetí jeřábu, nebo jeho částí do překážek a mimo prostor vyhrazený k pohybování jeřábu. Kolizní objekty jeřábu a jeho částí jsou zobrazeny na obrázku 43.



Obrázek 43: Kolizní objekty manipulačního jeřábu.

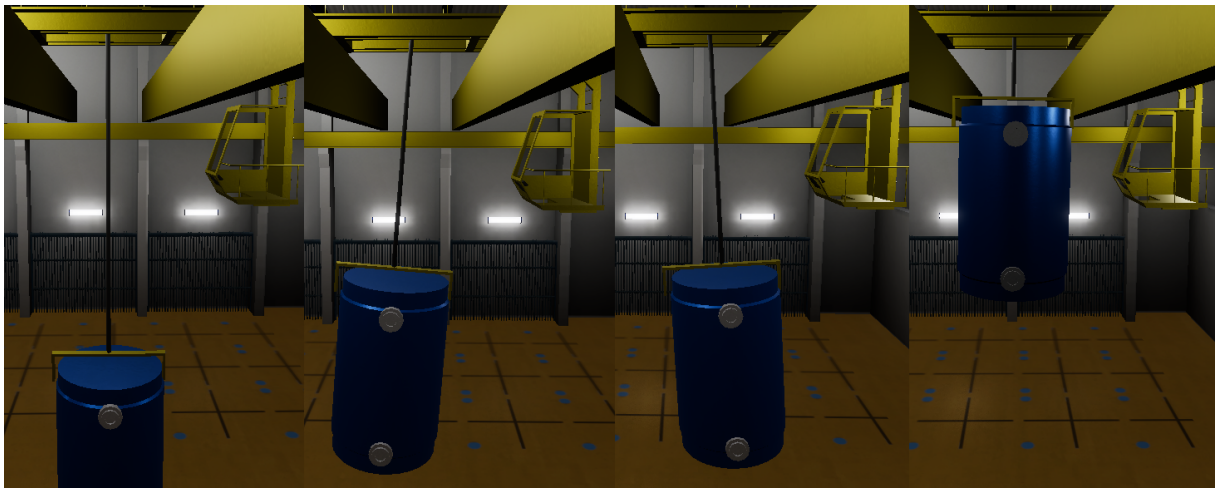
Pro zachytávání kolizí bylo využito funkcí pro transformace, které umožňují vracet výsledek kolize při jejich provádění (Sweep Hit Result). Struktura vracená transformačními funkce obsahuje výsledek kolize, pozici kde ke kolizi došlo, referenci na aktéra a jeho komponenty, se kterými objekt koliduje. Na základě této struktury kolize je ošetřen pohyb hlavních traverz jeřábu dopředu a dozadu, tak že na konci kolejnic jsou umístěny kolizní objekty, které představují zarážky. V případě, kdy uživatel vyvolá pohyb dopředu ale jeřáb v tomto směru koliduje se zarážkami na kolejnici není vstup uživatel brán v úvahu a jeřáb se nepohybuje. Stejným způsobem byly využity kolize i pro definování hranic pohyb plošiny jeřábu ve směrech doleva a doprava. Kolize byly využity i pro ošetření navíjení a odvíjení háku jeřábu. Začne-li kolizní objekt kolem háku kolidovat s kolizním objektem plošiny při pohybu háku nahoru, pohybování háku se přeruší. V opačném směru při odvíjení háku se kontrolují kolize háku s jinými kolizními objekty ve scéně na základě kterých se pohybování přeruší. V případě, že hák koliduje s objektem určeným pro přemisťování za pomoci jeřábu je uživateli nabídnuta možnost připnutí požadovaného objektu k háku jeřábu.

Kolize jednotlivých modelů jeřábu jsou řešeny tak, aby bylo možné přerušit i jiné směry pohybu než které jsou jim určeny. Například koliduje-li hák s nějakým objektem ve scéně při pohybu dopředu jsou přerušeny všechny pokusy uživatele o vyvolání pohybu v daném směru a uživatel tak musí upravit polohu háku nebo jiné části jeřábu tak, aby se překážce ve scéně vyhnul.

6.4.1 Přemisťování objektů

Jeřáb byl vytvořen za účelem přemisťování kontejnerů s použitým palivem ve skladu virtuální jaderné elektrárny. Model kontejneru je do scény umístěn jako aktor, který má aktivní kolizní objekt kolem sebe. Kontejner je ve scéně umístěn jako pohyblivý (Movable) objekt. Po najetí jeřábu na danou pozici a odvinutí háku nad kontejner, naznačí vyvolaná kolize háku a kontejneru, že je možné daný kontejner přichytit k háku. Pro přichytnutí a odpojení kontejneru od háku jeřábu byla využita funkce *Snap Actor To Component*, pomocí které lze připojit objekt ze scény ke komponentě jeřábu.

Po připojení kontejneru k háku jeřábu se na kontejner aplikují stejné metody jako na hák, čímž bylo docíleno korektního chování objektu při převozu. Například při pohybu jeřábu dopředu se společně s hákem pohyboval dopředu i kontejner připnutý k háku. Stejným způsobem byly obsluhovány i kolize mezi kontejnerem a ostatními objekty ve scéně. Příklad připojení kontejneru k háku jeřábu je zobrazen na obrázku 44.

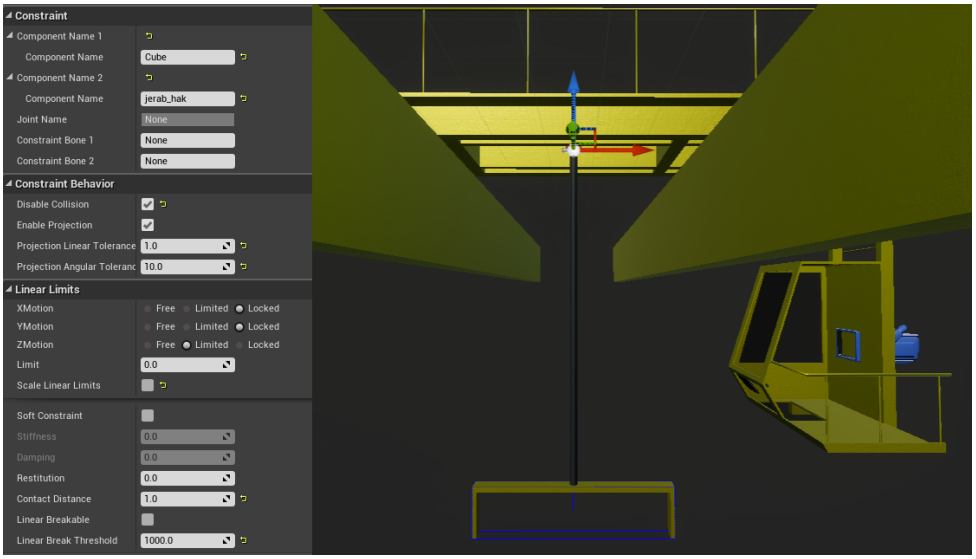


Obrázek 44: Připojení kontejneru k háku jeřábu.

7 Fyzikální vazby mezi objekty a jejich interakce

V této kapitole bude popisována technika pro spojování objektů ve scéně pomocí fyzické vazby mezi nimi, k čemuž v UE4 slouží komponenta *Physics Constraints*. Pomocí této komponenty lze ve scéně například přichytit lustr ke stropu. Obecně slouží komponenta ke spojení dvou aktorů ve scéně, kde alespoň jeden má zapnutou možnost simulace fyziky.

Převážně lze tuto komponentu využít pro modely, které jsou zpravidla na něčem „zavěšeny“. Při implementaci manipulačního jeřábu (viz kapitola 6) byla komponenta pro fyzickou vazbu mezi objekty použita pro zavěšení háku k plošině jeřábu.



Obrázek 45: Použití Physics Constraints ve scéně.

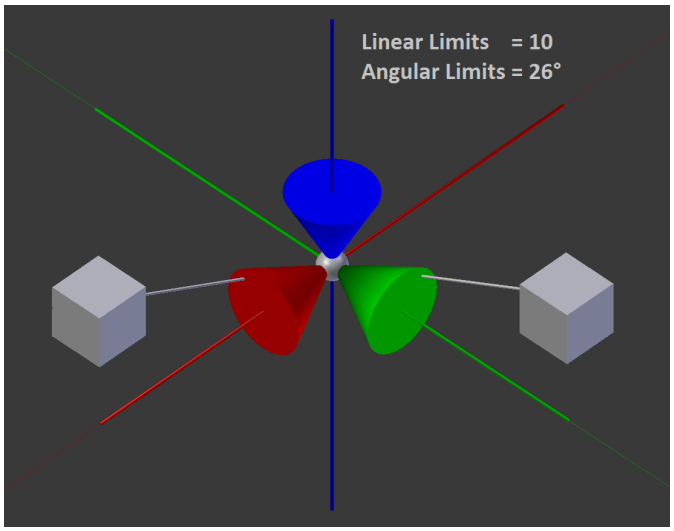
7.1 Možnosti nastavení Physics Constraints

Základním nastavením vazeb jsou reference na dané objekty, které mají být spojeny fyzikální vazbou. Nastavování požadovaných aktorů lze provést v záložce Constraint, kde jednotlivé reference jsou vkládány do *Constraint Actor*. Jednotlivým referencím lze zadat i specifické jméno v *Component Name* pomocí kterého se lze na dané aktory odkazovat ve skriptech a jiných aktorech ve scéně. V záložce chovaní vazeb pak lze vypnout kolize mezi spojovanými objekty. Podrobný popis nastavení jednotlivých možnosti a jejich dopad na objekty jsou popsány v dokumentaci UE [24].

7.1.1 Limity

Physics Constraints umožňuje nastavit limity spojení. Pomocí lineárních limitů lze ovlivňovat soudržnost objektů v závislosti na ose jejich dráhy pohybu. Pomocí XMotion, YMotion a ZMotion lze nastavit ve směru které osy a jakým způsobem bude ovlivňována vzdálenost mezi spojenými objekty. Hodnoty které lze nastavit jsou následující:

Volné (Free) - Žádné omezení v dané ose. Vzdálenost mezi objekty bude záviset jen na další nastaveních. Například na gravitaci, kdy objekty budou od sebe odtahovány i přesto, že jsou spojeny.
Limitované (Limited) - Omezení vzdálenosti mezi objekty v dané ose bude záviset na nastaveném limitu.
Uzamknuté (Locked) - Maximální vzdálenost mezi objekty v dané ose bude pevně daná umístěním ve scéně.



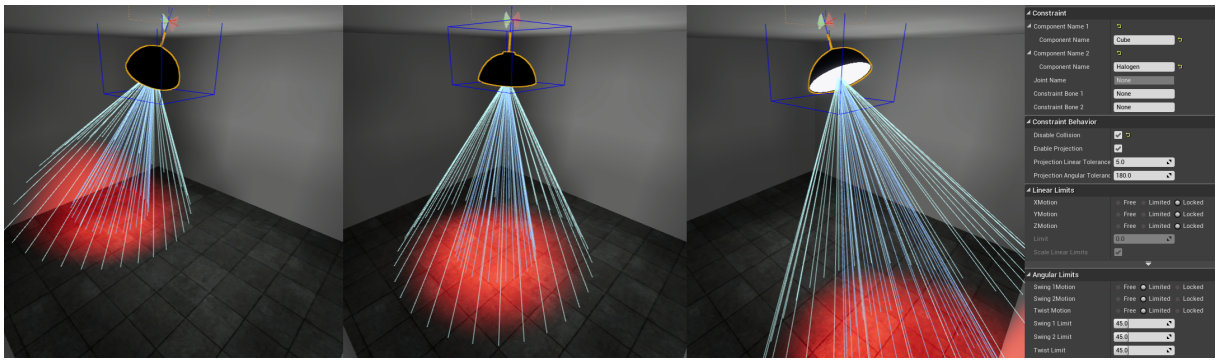
Obrázek 46: Ilustrace použitých limitů u *Physics Constraints* komponenty

Tytéž hodnoty lze nastavovat u úhlových limitů, kde se jimi ovlivňují nastavení *Swing Motion* a *Twist Motion*. Hodnoty nastavení těchto úhlových limitů udávají kolem které osy budou objekty rotovat v závislosti na bodě spojení. Použití limitů je ilustrováno na obrázku 46.

Dalším možností nastavení limitů je *Soft Constraint*. Toto nastavení lze provést jak u lineárních tak u úhlových limitů a nastavováním hodnot *Stifffness* a *Damping* lze ovlivnit tuhost a tlumení spoje mezi objekty. Dále pomocí *Linear/Angular Breakable* a nastavením *Linear/Angular Break Threshold*, pak lze nastavit mez přetrhnutí vazby na základě překročení mezní síly, respektive překročení nastaveného limitu. Pokud by pak na jeden z objektů působila síla překračující nastavenou mez, objekty se od sebe odpojí.

7.2 Příklad použití Physics Constraints

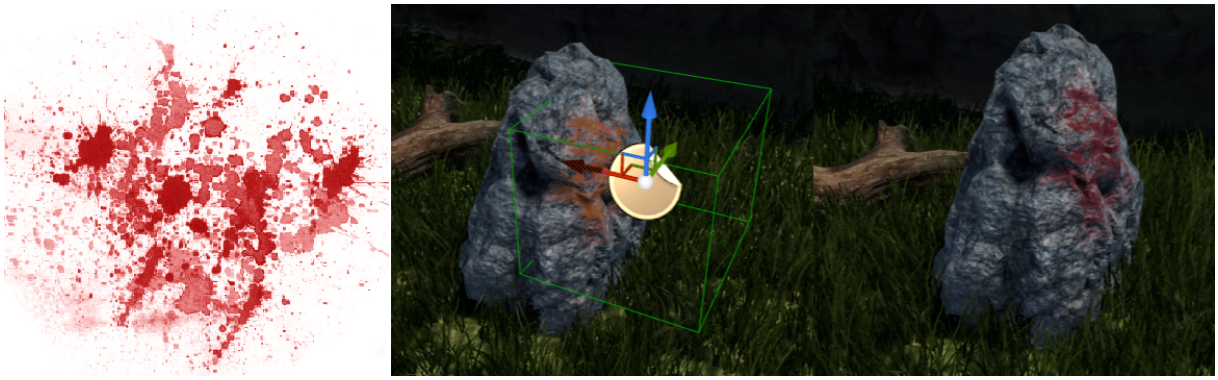
Komponenta Physics Constraints byla využita při vytváření manipulačního jeřábu ve skladu použitého paliva JE Dukovany. Konkrétně byl pomocí fyzikální vazby spojen hák jeřábu s plošinou. Původně byl hák umístěn staticky na danou pozici, ale pro navození reálnějšího dojmu při pohybování s jeřábem a zavěšeným objektem určeným k přepravě bylo nutné použít komponentu Physics Constraints. Názorná ukázka použití *Physics Constraints* a chování vazby při pohybu jeřábu je ilustrována na obrázku 45. Příklad použití Physics Constraints pro zavěšení světla ke stropu je pak ilustrován na obrázku 47.



Obrázek 47: Použití *Physics Constraints* komponenty pro zavěšení světla na strop.

8 Projekce textur a vlastností na objekty - Decals

Technika pro promítání textur a vlastností na povrch objektu ve scéně je využívána v počítačových hrách pro vytváření různých efektů. Například stopy po výstřelu ze zbraně, stopy krve, zašpinění zdí, nebo grafity na stěnách. Tato technika spočívá v promítání textury, respektive materiálu přímo na povrch objektu bez nutnosti zasahování do jeho topologie, nebo materiálu použitého pro daný objekt.



Obrázek 48: Decal Mesh - Projektor v Unreal Engine 4.

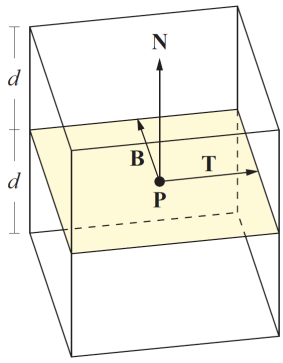
8.1 Algoritmus projekce textury

Technika Decals spočívá v mapování 2D obrázku na 3D povrch. Pro mapování textury se používá tzv. Decal Mesh označovaný jako projektor. Zpravidla se jedná o kvádr, který je přidán do scény a na základě jeho velikosti, polohy a orientace jsou pak získávány vrcholy a plochy povrchu objektu, který bude tímto projektorem ovlivněn. Na základě získaných fragmentů a jejich vrcholů je vytvořen nový povrch, na který se bude mapovat požadovaný materiál popřípadě textura. Ukázka projektoru v Unreal Engine 4 je zobrazena na obrázku 48. Algoritmus pro aplikaci decal efektu popisovaný v této části je podrobněji popsán Ericem Lengyelem v [25]

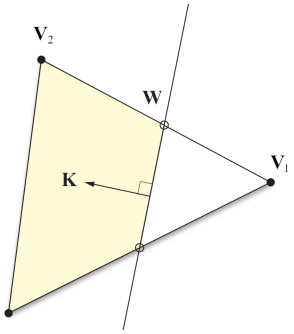
8.1.1 Projektor

Při výpočtu projektoru se vychází z bodu P, který je volen uprostřed projektoru. Dále je třeba definovat normály N, daného bodu. Bod P reprezentuje střed mapované textury. Například při zobrazování díry po výstřelu by tento bod představoval průsečík dráhy projektilu a povrchu objektu, na který bylo vystřeleno.

V bode P je pak potřeba znát i směr natočení, k čemuž slouží tangenta T a bitangenta B, kde $B = N \times T$. Bod P společně s normálou N a tangentou T pak tvoří orientovanou plochu.



Obrázek 49: Konfigurace projektoru.[25]



Obrázek 50: Ořezání polygonu.[25]

Ilustrace takto definovaného projektoru je zobrazena na obrázku 49. Jednotlivé stěny projektoru jsou pak definovány následujícími vektory:

$$left = \left(T, \frac{w}{2} - T \cdot P\right), \tag{11}$$

$$right = \left(-T, \frac{w}{2} + T \cdot P\right), \tag{12}$$

$$bottom = \left(B, \frac{h}{2} - B \cdot P\right), \tag{13}$$

$$top = \left(-B, \frac{h}{2} + B \cdot P\right), \tag{14}$$

$$front = (-N, d + N \cdot P), \tag{15}$$

$$back = (N, d - N \cdot P), \tag{16}$$

kde w představuje šířku a h hloubku projektoru. Vektory *left, right, top* a *bottom* definují stěny projektoru rovnoběžné s normálou N a *front* a *back* pak stěny kolmé k normále N. Hodnota d představuje polovinu výšky projektoru a označuje maximální hloubku vrcholu, který bude brán v úvahu pro projekci materiálu.

8.1.2 Vytvoření nového povrchu

Je-li definován tvar a pozice projektoru jsou následně hledány jednotlivé fragmenty resp. trojúhelníky zasahující, nebo ležící přímo v prostoru projektoru. Fragmenty ležící v projektoru lze hledat pomocí vzdálenosti fragmentu od orientované plochy definované bodem P a vektory B, T, N.

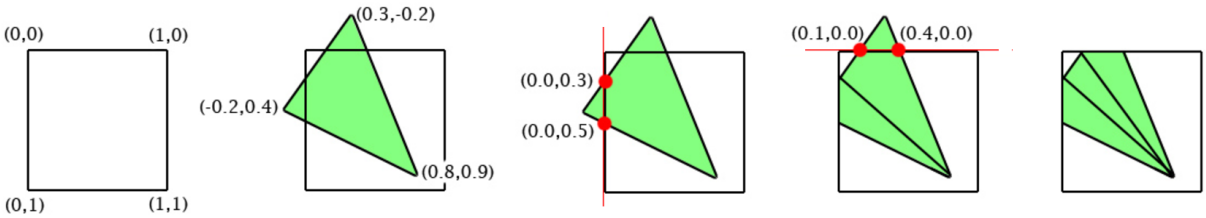
U každého nalezeného fragmentu se dále vyšetřuje zda jeho normála směřuje směrem definovaným normálou bodu P. Fragmenty, pro které platí:

$$N \cdot M < \varepsilon$$

,kde M je normála vyšetřovaného fragmentu a ε je pevně stanovená prahová hodnota, budou vyřazeny z dalšího zpracování, protože jsou odvráceny od promítací plochy procházející bodem P.

Na základě nalezených fragmentů je vytvořena trojúhelníková síť, která je vztažena k plochám definovaným vztahy 11 - 16. Přesahuje-li některý z nově vytvořených fragmentů stěny projektoru, jsou tyto fragmenty dále zpracovávány a v místě průniku ořezány.

8.2 Ořezání fragmentů



Obrázek 51: Zpracování polygonu přesahujícího hranice projektoru. [26]

Příklad, kdy fragment přesahuje hranice projektoru je ilustrován na obrázku 51. Před zpracováním jsou jednotlivé fragmenty transformovány ze světového souřadného prostoru (world space) do projekčního prostoru. V projekčním prostoru je pak projektor zobrazován jako čtverec (obrázek 51 - vlevo). Pokud leží vrcholy zkoumaného trojúhelníku mimo hranici projektoru (jejich souřadnice jsou mimo interval $\langle(0,0);(1;1)\rangle$), lze v místě kde se protíná hrana trojúhelníku a hrana projektoru vytvořit nový vrchol, který bude spojen hranou s výchozím vrcholem. Tento postup se opakuje dokud existují vrcholy, ležící mimo hranici projektoru. Následně jsou fragmenty transformovány zpět do světového souřadného systému a pro jednotlivé vrcholy jsou vypočítány texturovací souřadnice, na základě kterých je mapována textura použitého materiálu pro Decal.

8.2.1 Výpočet texturovacích souřadnic

Posledním krokem při vytváření Decal efektu je výpočet texturovacích souřadnic pro jednotlivé vrcholy tvořící povrch, na který bude projektován materiál, popřípadě textura. Na základě vzdálenosti mezi jednotlivými vrcholy trojúhelníkové sítě Q a promítací plochy projektoru procházející bodem P, jehož orientace je dána vektory T, B, N lze pro výpočet texturovacích souřadnic odvodit následující vztahy:

$$s = \frac{T \cdot (Q - P)}{w} + \frac{1}{2}, \tag{17}$$

$$r = \frac{B \cdot (Q - P)}{h} + \frac{1}{2}. \tag{18}$$

8.3 Tvorba Decals v Unreal Engine 4

Základem promítání Decal v UE4 je materiál s doménou *Deferred Decal*, vytvořený přímo pro účely použití v Decal projektoru. Decal projektor s takto definovaným materiálem lze pak do scény umístit manuálně, nebo jej lze vytvářet pomocí volané funkce na základě definované akce. Například při výstřelu projektilu do zdi se vytvoří projektor mapující texturu odpadlé omítky na místě, kde se dráha projektilu protíná se zdí. Výsledek projekce Decal projektoru je závislý na dalších nastavení použitého materiálu, použitých texturách a koeficientech. Mezi další nastavení materiálu patří tzv. režimy prolnutí (Blend Modes), které do značné míry ovlivňují výslednou projekci a barvu.

8.3.1 Režimy prolnutí (Blend Modes)

U Decal materiálů vytvářených v UE4 lze nastavit několik režimů prolnutí barvy definované materiálem a barvy, která je na povrchu objektu, na který je Decal promítán. Jednotlivé režimy pak ovlivňují vlastnosti a vstupy, které lze u materiálu definovat.

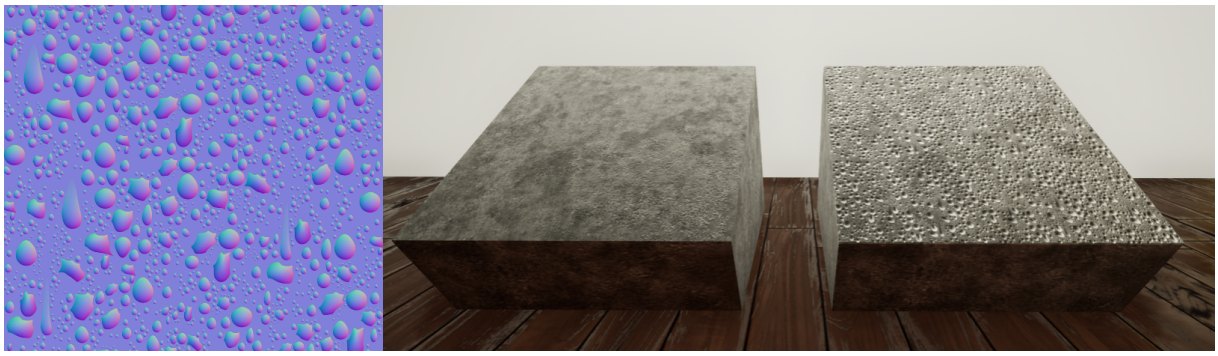
Translucent: Při nastavení prolnutí výsledné barvy na možnosti Translucent lze u projektovaného materiálu nastavit difuzní, metalickou, zrcadlovou a emisní složku výsledné barvy. Dále lze nastavovat drsnost povrchu a průhlednost. Výsledná barva promítaná na povrch nějakého objektu pak odpovídá barvě definované v Decal materiálu. Vlastnosti materiálu použitého na povrch, na který je Decal projektován jsou potlačeny. Příklad použití Translucent Decal materiálu implementovaného v UE4 je zobrazen na obrázku 48.

Stain: Nastavením prolnutí na režim Stain, je výsledná barva Decal efektu ovlivněna jak materiálem samotného objektu tak materiálem použitým v projektoru. Decal zachovává vlastnosti původního materiálu a výsledná barva je dána smícháním obou použitých materiálů. Možnosti

nastavení Decal materiálu a jeho vstupy jsou stejné jako u režimu Translucent. Použití Stain Decal materiálu je zobrazeno na obrázku 52.

Obrázek 52: Stain Blend Mode.

Normal: Tento režim umožňuje Decal materiálu nastavit pouze normálovou mapu a průhlednost použité normálové textury. Výsledkem projekce takto nastaveného materiálu na povrch objektu je změna normálové mapy původního materiálu objektu, bez ovlivnění jeho ostatních vlastností (diffuse, specular, metalic apod.) Příklad použití Normal Decal materilálu pro vytvoření efektu kapek deště na kovovém povrchu je zobrazeno na obrázku 53.

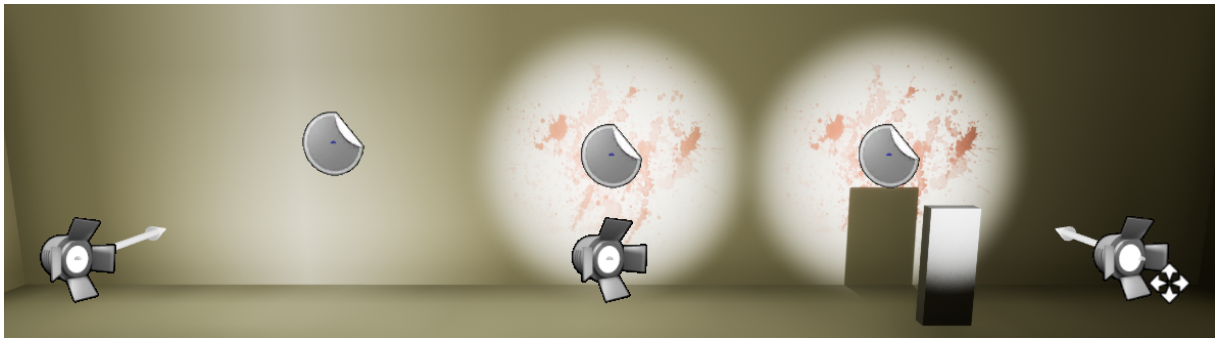


Obrázek 53: Normal Blend Mode.

Emissive: Režim prolnutí typu Emissive umožňuje Decal materiálu nastavit jen emitovanou barvu a průhlednost. Tyto vlastnosti jsou pak projektorem promítány na povrch objektu, kde výsledná barva je skládána z vlastností původního materiálu a emitující barvy použité v Decal materiálu. Tento režim lze použít například pro vytvoření světelné emise ze zdroje světla.

Výše popisované režimy patří mezi základní, které lze defaultně nastavit u Deferred Decal materiálů. Během implementace a použití Decals s těmito režimy prolnutí se vyskytl problém, kdy jednotlivé Decals se renderovali jen pokud byly přímo nasvětleny pohyblivým, nebo stacionárním světlem, protože tyto světla jsou počítána pomocí tzv. dopředného vykreslování (Forward Rendering). Stejným způsobem jsou počítány i Decals se základními režimy prolnutí. Důsled-

kem toho bylo, že Decals, které byly ve scéně překryty stínem, popřípadě nebyly v dosahu nějakého dynamického světla, se nezobrazovali korektně. Názorný příklad je zobrazen na obrázku 54, kde jsou na Decal s režimem prolnutí Translucent použity tři světelné zdroje (statický, stacionární a dynamický). Teto jev je zapříčiněn metodou zpětného vykreslování (Deferred Rendering), ve kterém se počítají statické světla a stíny. Na základě vytvořené Lightmapy, do které nejsou zahrnuty Decals se základními režimy prolnutí, se pak hodnota původně vypočtené barvy přepíše hodnotou vypočteného osvětlení. Ve stínu je pak namísto promítaného Decal materiálu zobrazován původní materiál objektu, jehož jas je redukován vypočteným stínem. Pro korektní zobrazování promítaného materiálu na povrch objektu je potřeba použít tzv. Decal Buffer a Decal materiálu nastavit odpovídající režim prolnutí.

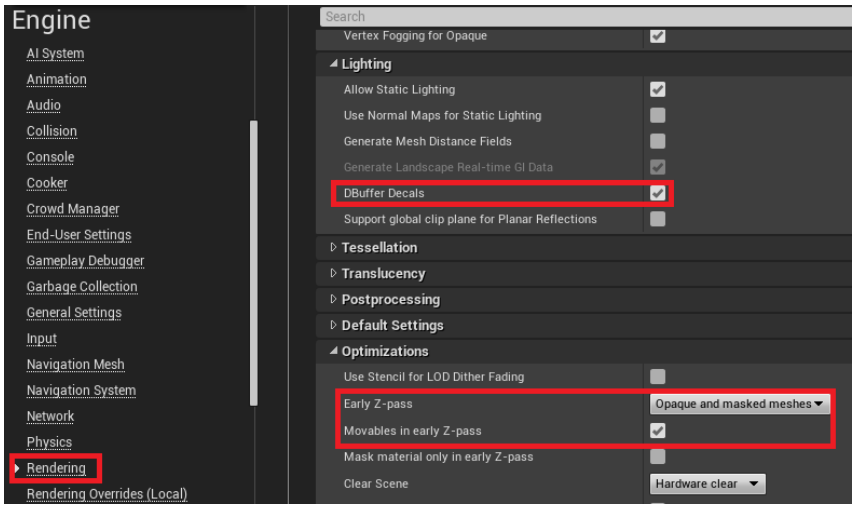


Obrázek 54: Vykreslování Decals v závislosti na zdroji světla (Static, Stationary, Movable).

8.3.2 DBuffer Blend Modes

Pro korektní zobrazování Decal i na zastíněných plochách objektu nebo ve statickém světle, lze použít režimy prolnutí využívající Decal Buffer, do kterého jsou ukládány vypočtené hodnoty promítaného Decal materiálu a pomocí tohoto bufferu jsou tak k dispozici při zpětném vykreslování a vytváření lightmapv místě, kde je Decal promítán.

Využití DBufferu pro zobrazování Decals ovšem není standardně podporováno UE4 a musí být povoleno v nastavení projektu. V *Project Settings* v záložce *Rendering* je potřeba v sekci *Lighting* aktivovat nastaveni *DBuffer Decals*. Dále je potřeba v sekci *Optimization* zkontrolovat nastavení *Z-pass*, které by mělo odpovídat nastavení zobrazenému na obrázku 55. Unreal Engine bylo po tomto nastavení třeba restartovat.

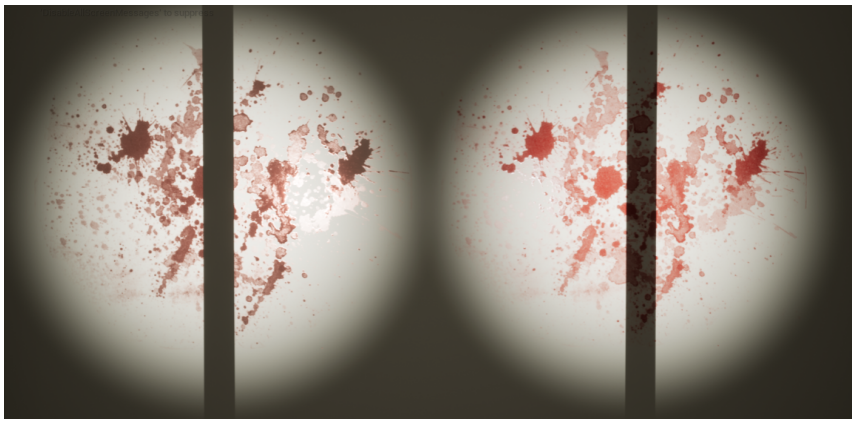


Obrázek 55: Aktivace Decals Bufferu v nastavení projektu UE4.

Po zapnutí DBufferu bylo možné využít další režimy prolnutí v nastavení Decal materiálu. Použití těchto režimů zamezuje nežádoucímu překreslení promítaného materiálu lightmapou a aplikovaný Decal je zobrazován korektně při použití dynamického i statického osvětlení. Jednotlivé režimy Decal materiálu využívající pro své zobrazování DBuffer, jsou následující:

- DBuffer Translucent Color, Normal, Roughness
- DBuffer Translucent Color
- DBuffer Transculent Color, Normal
- DBuffer Transculent Color, Roughness
- DBuffer Transculent Normal
- DBuffer Transculent Normal, Roughness
- DBuffer Transculent Roughness

Jednotlivé vlastnosti Decal materiálu, které lze použít při daném režimu jsou patrné z názvu použitého režimu. Rozdíl ve zobrazování Decals materiálu s režimem prolnutí Translucent, a Decals materiálu s režimem prolnutí DBuffer Translucent je zobrazen na následujícím obrázku.



Obrázek 56: Zobrazování Decals: Translucent (vlevo) DBuffer Translucent (vpravo).

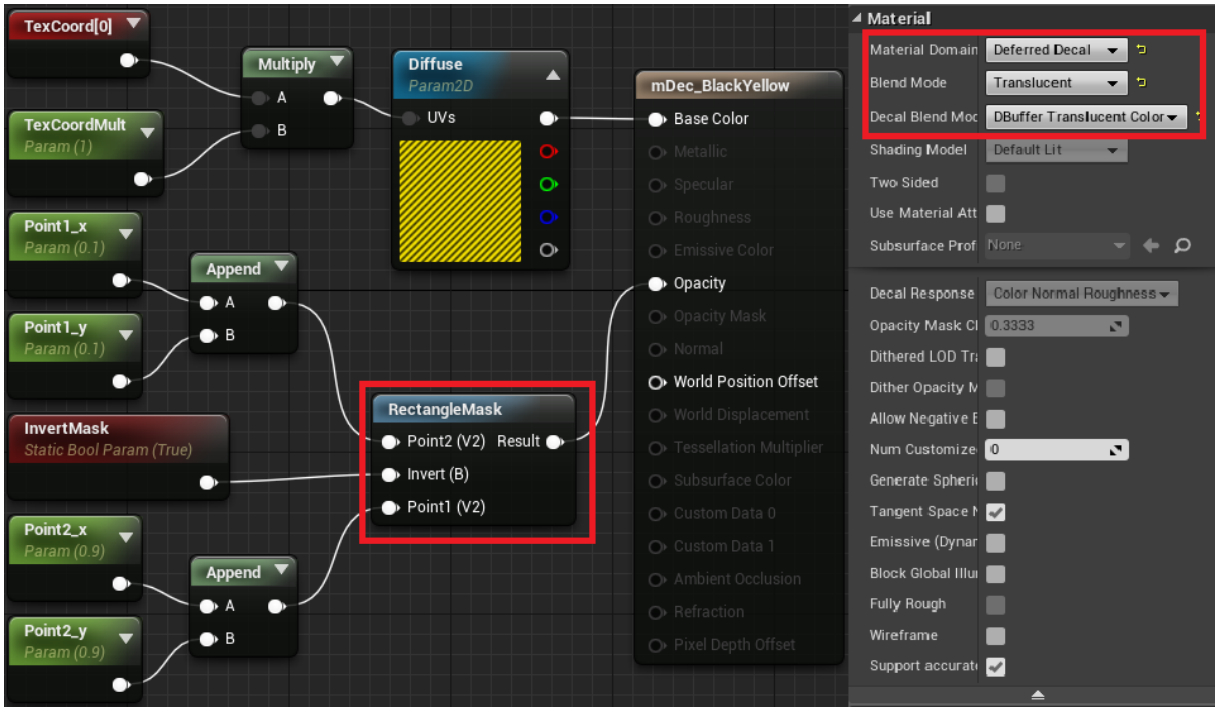
8.4 Příklad implementace Decals

Během tvorby projektu pro vizualizaci provozu jaderné elektrárny bylo potřeba vytvořit objekt, který by se dal použít pro vizualizaci výstražných bezpečnostních nátěrů (viz obrázek 57).

Obrázek 57: Výstražné barevné značení ve strojovně JE Dukovany.

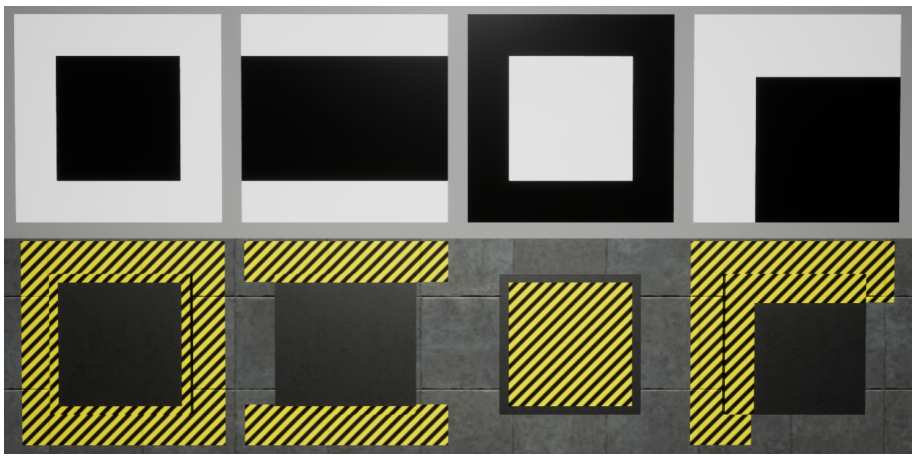
Pokud by byly textury pro bezpečnostní nátěry mapovány klasickým způsobem, bylo by potřeba upravit topologii a texturování přímo při vytváření modelu. Pomocí Decal lze na objekt promítat texturu bez nutnosti upravovat jeho topologii a materiál.

Pro promítání textury byl vytvořen Decal materiál s režimem prolnutí DBuffer Translucent Color, protože v materiálu bude použita pouze difusní textura. Vytvořený materiál a jeho nastavení je zobrazeno na obrázku 58. Vzhledem k tomu, že textura by měla být mapována v okolí



Obrázek 58: Ukázka nastavení Decal materiálu (Shaderu).

nějakého objektu (například šachta, dveře apod.), bylo potřeba pomocí masky průhlednosti nastavit čtvercový výřez, kde bude textura průhledná a nebude tak ovlivňovat barvu povrchu na který bude promítána. Jako masku by bylo možné použít texturu, ale velikost čtverce by nebylo možné měnit a závisela by na použité textuře. Pro použití jiné velikosti čtverce nebo použití jiných velikostí jeho hran by bylo nutné vytvářet a importovat nové textury.



Obrázek 59: Důsledek použité masky na promítaný Decal.

Proto bylo potřeba vytvořit funkci, která na základě zadaných parametrů vygeneruje čtvercovou masku požadovaných rozměrů přímo za běhu aplikace. K implementaci dané funkce byl využit tzv. Material Function editor, sloužící k vytváření funkcí volaných z materiál editoru. Funkce vytvořené pomocí Material Function Editoru je pak možné volat ve všech vytvořených materiálech bez ohledu na jejich kategorizaci. Volání implementované funkce *RectangleMask* v materiál editoru je znázorněno na obrázku 58. Funkci jsou předávány parametry určující pozici dvou řídících bodů čtverce resp. obdélníku z intervalu $\langle(0,0);(1,1)\rangle$ a hodnota určující zda má být výsledná maska invertována. Na základě vstupních parametrů je pak vygenerována binární maska. Příklad generované masky a její vliv na promítanou texturu je zobrazen na obrázku 59. Praktické využití promítání dané textury v projektu vizualizace provozu jaderné elektrárny je zobrazeno na obrázku 60.

Obrázek 60: Příklad použití Decal ve scéně - strojovna virtuální jaderné elektrárny.

9 Závěr

Během realizace této práce bylo vytvořeno několik scén, na nichž jsou demonstrovány metody a techniky popisované v rámci této práce. Jednotlivé scény jsou zahrnuty v ukázkové aplikaci, která je součástí praktického výstupu této práce. Zpracované modely a některé scény jsou také využity v projektu pro vizualizaci provozu jaderné elektrárny. Jedná se o modely budov kontejnmentu, strojovny, skladu použitého paliva a některých budov s podpůrnými systémy. Tyto modely jsou součástí venkovní scény v projektu JE, na jejímž ztvárnění pracovalo více studentů.

Součástí projektu virtuální JE jsou dále jednotlivé scény, vytvořené v rámci praktického zpracování této práce, na nichž jsou vizualizovány vnitřní prostory některých zmíněných budov. V rámci většího prohloubení znalostí z prostředí herních enginů, lze téma této práce dále rozšířit o studium konkrétních technik používaných při implementaci daného typu scény. Během praktického zpracování se ukázalo, že způsoby použité při vytváření statických scén se do jisté míry liší od způsobů a technik aplikovaných při realizaci dynamické scény, nebo scény určené pro zobrazování ve virtuální realitě, kde výběr použitých metod má velký vliv na výpočetní výkon a rychlost zobrazování vizualizované scény.

V rámci praktické části této práce byly dále vytvořeny dva univerzální moduly (plug-in). Jeden pro vytváření a vizualizaci mapy zobrazované scény. Druhý pro nastavování úrovní grafických možností aplikací vytvářených v prostředí Unreal Engine 4. Moduly byly implementovány tak, aby je bylo možné využívat v různých projektech UE a jejich použití bylo jednoduché a srozumitelné. V rámci testování byly plug-in moduly nabídnuty k odzkoušení studentům předmětu Modelování v grafických aplikacích a jsou součástí projektů vytvářených v rámci jiných diplomových a bakalářských prací. V rámci dalšího vylepšování je možné obohatit implementované programy o vizuálně kvalitnější zpracování uživatelského rozhraní a jednotlivé moduly rozšít o nové funkce.

Přínos této práce je především v nastínění postupů, využitých při implementaci projektů zaměřených na realistickou vizualizaci, které čtenář může využít při vlastním vývoji grafických aplikací v prostředí Unreal Engine 4 a nabyté znalosti tak dále prohlubovat.

Literatura

[1] DELOURA, Mark. In: *GameEngines and Middleware in the West* [online]. 2011 [cit. 2017-03-03]. Dostupné z: https://www.slideshare.net/markdeloura/game-engines-and-middleware-2011

[2] Epic Games. *Unreal Engine*[online]. [cit. 2017-01-03]. Dostupné z: https://www.unrealengine.com/what-is-unreal-engine-4

[3] Documentation. *Unreal Engine 4* [online]. [cit. 2017-01-29]. Dostupné z: https://docs.unrealengine.com/latest/INT/

[4] *Unreal Engine* [online]. [cit. 2017-03-10]. Dostupné z: https://www.unrealengine.com/

[5] Unity. *Game Engine* [online]. [cit. 2017-02-23]. Dostupné z: https://unity3d.com/

[6] Company Facts. *Unity* [online]. [cit. 2017-02-23]. Dostupné z: https://unity3d.com/public-relations

[7] *CryEngine*[online]. [cit. 2017-02-23]. Dostupné z: https://www.cryengine.com/

[8] *Steam* [online]. [cit. 2017-01-13]. Dostupné z: http://store.steampowered.com/

[9] Source Engine. *Valve Corporation*[online]. [cit. 2017-01-13]. Dostupné z: http://www.valvesoftware.com/index.html

[10] Vulcan. *Kronos* [online]. [cit. 2017-01-13]. Dostupné z: https://www.khronos.org/vulkan/

[11] Physically Based Rendering. *Coding Labs* [online]. [cit. 2017-03-30]. Dostupné z: http://www.codinglabs.net/article_physically_based_rendering.aspx

[12] VRIES, Joey. Learn Opne GL: *PBR Theory* [online]. [cit. 2017-03-30]. Dostupné z: https://learnopengl.com/#!PBR/Theory

[13] KAUTZ, Jan a Sumanta PATTANAIK, ed. *Microfacet Models for Refraction trough Rough Surface* [online]. In: . 2007 [cit. 2017-03-30]. Dostupné z: https://www.cs.cornell.edu/~srm/publications/EGSR07-btdf.pdf

[14] KARIS, Brian. *Real Shading in Unreal Engine 4* [online]. 2013 [cit. 2017-04-01]. Dostupné z: http://blog.selfshadow.com/publications/s2013-shading-course

[15] *Blender* [online]. [cit. 2017-04-25]. Dostupné z: https://www.blender.org/

[16] HOPPE, Hugues. *Progreassive Meshes* [online]. [cit. 2017-04-01]. Dostupné z: http://hhoppe.com/pm.pdf

[17] *Hitman* [online]. [cit. 2017-03-05]. Dostupné z: https://hitman.com/

[18] *Aliens vs Predators* [online]. [cit. 2017-03-05]. Dostupné z: http://www.aliens-vs-predator.cz/

[19] Command-Line Arguments. *Unreal Engine* [online]. [cit. 2017-03-21]. Dostupné z: https://docs.unrealengine.com/latest/INT/Programming/Basics/CommandLineArguments/index.html

[20] Hardwarový a softwarový průzkum. *Steam* [online]. [cit. 2017-01-03]. Dostupné z: http://store.steampowered.com/hwsurvey

[21] Scalability Reference. *Unreal Engine*[online]. [cit. 2017-04-21]. Dostupné z: https://docs.unrealengine.com/latest/INT/Engine/Performance/Scalability/ScalabilityReference/

[22] SOJKA, Eduard. *Počítačová grafika II*: průvodce studiem. Ostrava: VŠB - Technická univerzita, Regionální centrum celoživotního vzdělávání, 2003. ISBN 80-248-0293-7.

[23] Moving Physical Objects. Unreal Engine [online]. [cit. 2017-04-03]. Dostupné z: https://www.unrealengine.com/blog/moving-physical-objects

[24] Physics Constrainsts. Unreal Engine [online]. [cit. 2017-04-03]. Dostupné z: https://docs.unrealengine.com/latest/INT/Engine/Physics/Constraints/

[25] LENGYEL, Eric. Mathematics for 3D game programming and computer graphics. 3rd ed. Boston, MA: Course Technology, Cengage Learning, c2012. ISBN 14-354-5886-9.

[26] ROSEN, David. Project Decals. In: Wolfire Games Blog [online]. 2009 [cit. 2017-04-03]. Dostupné z: http://blog.wolfire.com/2009/06/how-to-project-decals/

[27] DUTRÉ, Philip, Philippe BEKAERT a Kavita BALA. *Advanced global illumination*. Natick: A K Peters, c2003. ISBN 1-56881-177-2.

Seznam příloh

Obsah přiloženého CD

Dokument v elektronické podobě.

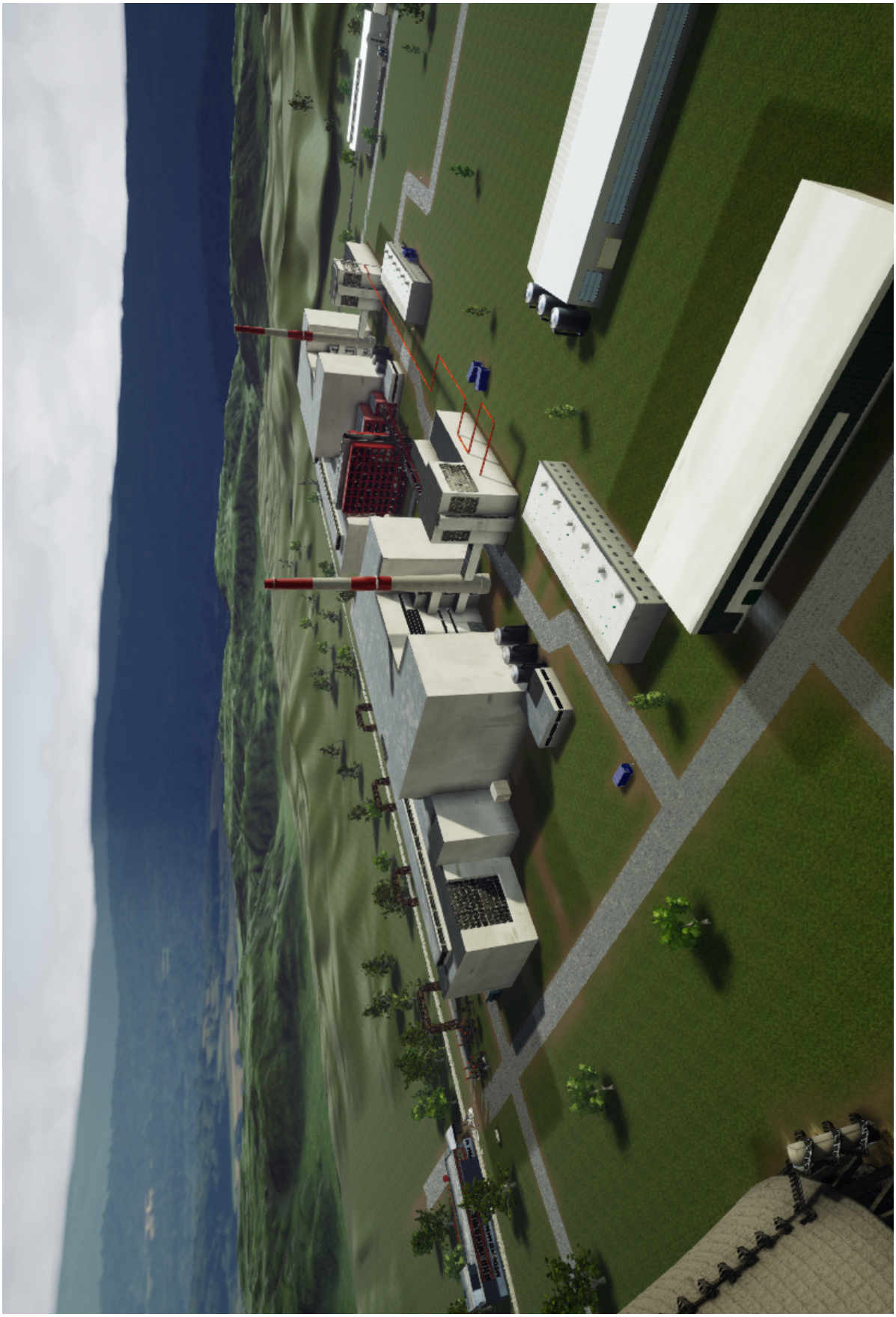
Plugin pro generování map virtuálního světa.

Plugin pro menu nastavení grafických možností.

Ukázková aplikace.

A Vytvořený model ve venkovní scéně JE - pohled 1

B Vytvořený model ve venkovní scéně JE - pohled 2



C Vytvořený model ve venkovní scéně JE - pohled 3

D Scéna zobrazující teoretický model JE

E Scéna zobrazující vnitřní prostory primárního okruhu JE